Yesterday TODAY

21

Captain COMAL
Buries BASIC

Cover Feature
of issue #8

The Amazing Adventures of CAPTAIN COMAL™

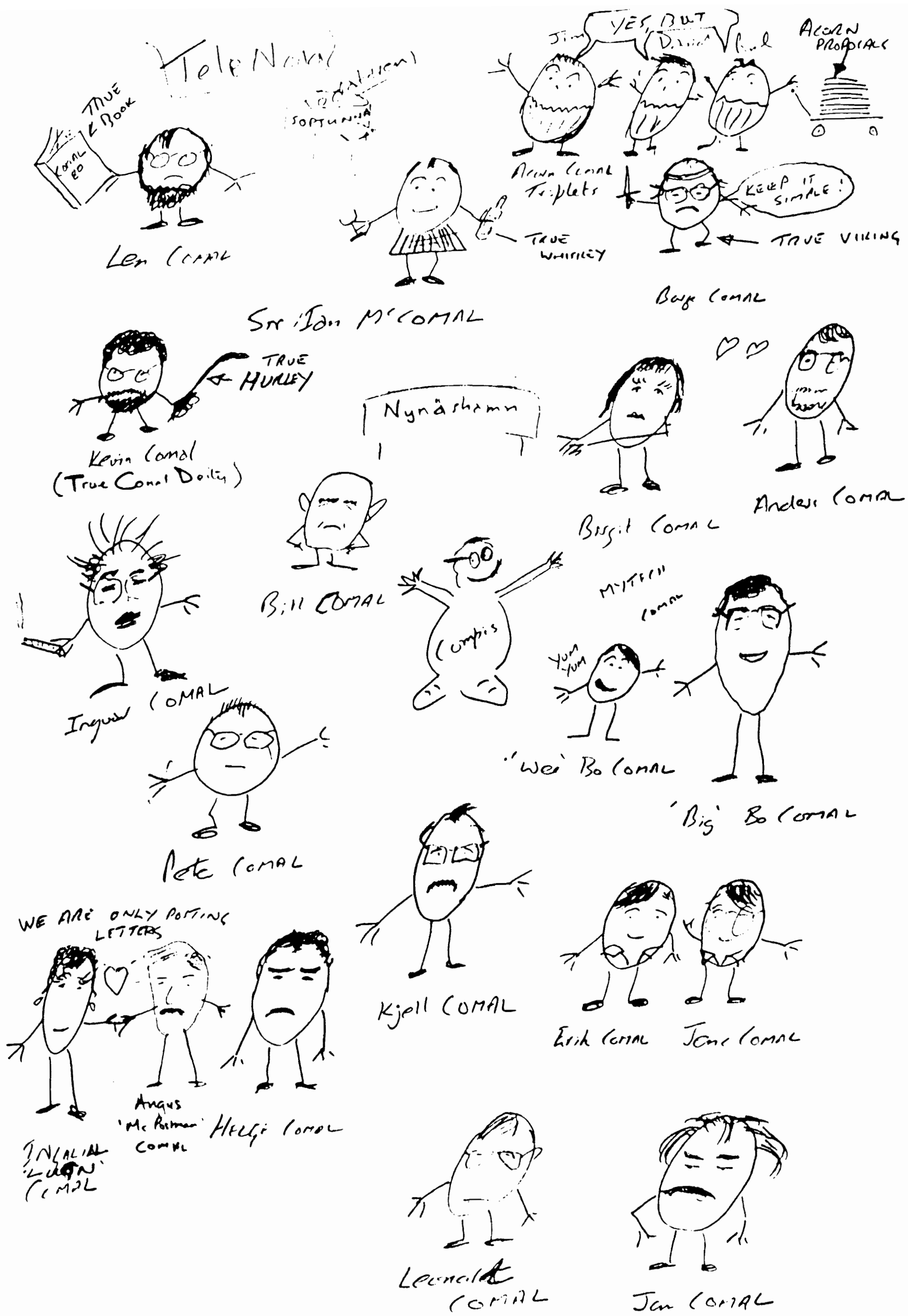
Calvin the
COMAL Turtle

Cover Feature
of issue #5

From the cover of issue #17
Draw Poker Card Game

From the cover of issue #12
Benchmark Timing Tests

From the cover of issue #18
Mouse Package


TRUE
Len Comal
Sir Ian McCOMAL
TRUE WHITLEY
YES, BUT
Jim
David
Paul
ACORN PROPOSALS
Acorn Comal Triplets
KEEP IT SIMPLE!
TRUE VIKING
Borge Comal
TRUE HURLEY
Kevin Comal
(True Comal Deity)
Nynäshamn
Birgit Comal
Anders Comal
Bill COMAL
Compis
MYTECH
Comal
Yum Yum
Ingvar COMAL
Pete Comal
'Wee' Bo Comal
'Big' Bo Comal
WE ARE ONLY POTTING LETTERS
Angus 'McPostman' COMAL
Helgi Comal
Kjell COMAL
Erik Comal
Jens Comal
Leonard COMAL
Jan COMAL

Editor's Notes

COMAL Users Group, U.S.A, Ltd. started as a one man operation, and now is back to that. I will need 2 extra months to adjust to the new small office and no staff. This is the perfect time for a special issue to look back over four years of *COMAL Today*. This is a light hearted look at the Yesterday of *COMAL Today*.

New readers may not be familiar with the original representation of "*Captain COMAL*". A Danish Artist drew our leader as a "*circle man*". He did a series of pictures, including the one on the cover (*from issue #8*), showing the "*captain*" burying BASIC. The inside cover pokes fun by depicting each of the people at the Standards meeting as *circle people*. Recently, we made "*Calvin the COMAL Turtle*" our mascot. He has since been seen on T-Shirts, Doc Boxes, disk labels, and even *COMAL Today* covers.

We start our "*look back over Today*" on the next page, beginning with issue #1. While **Programming Languages For Beginners** may look primitive, it is one of my favorite articles. It gives an interesting perspective direct from COMALs founder... and shows how we have improved the look of our pages!

You may wish to compare the **Print File** article from issue #2 with the **Files** article in issue #20. Note that it includes a mistake; **test$** should be **temp$**. Next, Colin reminds us of the initial struggle we had with COMAL.

Error Messages Built Into COMAL is a reminder to newcomers that originally, COMAL 0.14 had to access a disk file for its error messages! The following **Strings In COMAL** from issue #5 points out how much important information was in those early issues.

Cheer Up - It Could Be Worse. I couldn't stop laughing each time I read it. Maybe you had to be at the show to appreciate the article. Who knows. Next we hired Debra Ruth Junk to help improve the look of *COMAL Today*. She also provided a non-computerist view in **Illiterates Unite** in issue #7, followed by **The Truth About Computers** in #8.

Our look did improve. Acquiring our Laser Printer was a big step, though when we used it in issue #9, we only could manage elite style type. **Rod The Roadman** is an example of Borge Christiansen's expanding vision. That issue also saw the introduction of our *racing stripe* continuation headers. You can see them on top of the second page of **Comparing Disk Files**.

Text Package was just one of the advances provided by the Dutch Users Group. It also was our attempt to list a package itself in *COMAL Today*. **Instructional Videos** presents an interesting idea. It also shows our disk and cartridge *icons* (*drawn on a MacIntosh*) we used to categorize articles. **VAL and STR$ 0.14** reminds us of the struggles we had before **Power Driver** built them into the system!

Kevin Quiggle presents our first **Fractals** program in issue #12 (*he also painstakingly indexed the first 12 issues*). Next, remember the revelation that the cartridge included **Integrated Software**? The **While Loops** article is just one in a series of articles that explain COMAL fundamentals to beginners. Then **Program Outliner** shows how easy it is for COMAL to outline a program listing. $97 indeed!

Issue #15 saw our massive presentation of ready to use procedures, along with David Stidolph's **Introduction to Procedures**. Then David explains our surprise while testing his pixel flipping. This also was the issue where our Laser printer finally produced camera ready pages for us, complete with headers, footers, and even the **more»** reminder at the bottom corner.

Next, Jack Baldrige explains how you can spend 14 hours to **Calculate PI** to 2500 places! We conclude our *Look Back* with Robert Ross's **Guest Editorial** and the introduction of *Buzz*. ■

----- PROGRAMMING LANGUAGES FOR BEGINNERS ----- AND THE GLOBAL CHALLENGE

by Borge Christensen, COMAL founder
(reprint from COMAL CATALYST #2 with permission)

The first time I heard the word "basic" was in 1946. I met a fisherman one day who told me that he had just returned from England. It appeared that he was one of the many Danish seamen who had escaped to England shortly after Denmark was invaded in 1940, and had come to serve in the English navy. At the time, when I met him, I had been learning English in school for about a year, and I was eager to know, how it was to come to England just like that. What about the language! Did he speak English when he came over? Or how did he learn it? He answered my many questions by telling me that they were all offered a course in something called "Basic English". As I asked him, what that was like, he gave me an answer that I did not fully understand then. "It was a whore of a language. It could be used to cover your immediate needs, but there was no real pleasure in it".

A NEW MIRACLE ----- Soon after that the word entered my English vocabulary in its general meaning, and it went on to be just another common, useful, decent English word until 1971. At that time our mathematics department got a minicomputer installed. A Data General Nova 1200. A new miracle. We were all very proud and it was even mentioned in the local newspaper. Since I had taken an Algol course at the university, and was therefore considered to be the local expert - nobody knew that the course had only been a very short one, and that I remembered very little about Algol or computers in general - I was asked to take care of the "recent device" and maybe even teach some of the students to use it. And then I met the word "BASIC" again in a more specific context. And spelt with capitals! After a year and several "miles" of BASIC programs, I realised that something was rotten in the state we were in. Very often I found it quite hard to find out what was going on in the students programs, and especially the faulty ones, of course.

DEFICIANCIES ----- At first I blamed my own lack of profound knowledge for the whole misery, but gradually I dared to think that this marvel of a new language - that had come from THE STATES - might have some deficiencies. My suspicions were confirmed as I started to talk about it to some of my colleagues at the department of informatics, university of Aarhus. They told me straightforward that BASIC was disaster, and that its success was totally undeserved and due to the fact that no other language was generally available on the small computers used in elementary education. One of them, Benedict Lofstedt, took real interest in my problem - it is often very difficult for a teacher to get university reasearchers interested in educational problems - and he advised me to read a book that had just come out. Its title was SYSTEMATIC PROGRAMMING, and it was written by one Niklaus Wirth.

HUMAN INTERFACE ----- I bought it, started to read it, and after a month I knew that the folks from Aarhus were right. BASIC IS a disaster and to a very large extent to blame for the bad programming that was going on not only in our place, but in most other colleges and highschools. But what could be done? Pascal - the language submitted in SYSTEMATIC PROGRAMMING - was only implemented on few large mainframes and used exclusively in academic circles. In those days very few people anticipated the status Pascal has today. And after all, the language is not all there is to it. The ENVIRONMENTS of the language are of crucial importance, too. What we today might call "human interface". The Basic on our computer - DG XBASIC - was fully interactive, supported by a good multi-user system, and included very useful file handling. We could not do without all that.

After having considered all this, I came back to Benedict Lofsted and he now suggested that we design a few but powerful extensions to the language and system we had, and that we should use as much of the ideas from Pascal as possible. It was quite obvious, what was most urgent: LONG variable names, a GLOBAL IF-THEN-ELSE structure (NOT the one line IF-THEN-ELSE that comes with any silly BASIC nowadays), REPEAT- and WHILE- loops, a multi-branching CASE structure, and NAMED subroutines. During the following six months the design was finished. I coined the work COMAL (COMmon Algorithmic Language) to name the extensions we had planned, and in June 1974 we started to implement the facilities, mentioned above, on the Nova computer her in Tonder. The first versions of COMAL were launched in February 1975. We were now able to write programs like this:

```
IF TRY<3 THEN
  PRINT "NO, TRY AGAIN"
ELSE
  PRINT "NO, THE CORRECT ANSWER IS";RESULT
  PRINT "TYPE THAT."
ENDIF
```

I have chosen this example because the displayed IF-THEN-ELSE structure was the one that made COMAL popular very fast. The early versions of COMAL allowed IF-THEN-ELSE branches to be nested to a depth of only four, but it is not very often you need more, and it is much easier to do it the COMAL way that by applying GOTO statements. You do not have to go much deeper than two levels to get BASIC programs that are at least hard to read, and most beginners will get lost in the labyrinth of BASIC statements that implement more than three nested branchings. I ought to mention that the line indentation shown in the example is done automatically by the COMAL interpreter.

IMPACT OF COMAL ----- The impact of COMAL has been stronger than we had guessed that it would be. The students write much better programs in COMAL than they used to do in BASIC, and - even more important - their programs have become

READABLE. At the beginning our ideas were rejected by quite a lot of teachers. In general students were much faster to see that a good tool had come into their hands. Sometimes when I came out to talk about COMAL I got the impression that I had started some kind of a religious war and not just invented some useful improvements of a programming language that might be discussed in terms of expediency and effectiveness. Gradually the attitude changed in favor of the concept, however, and since 1977/78 it has become very difficult to sell a BASIC computer to a school in Denmark. Most teachers will ask the salesman to come back again some other day with a computer that can run COMAL.

A NEW VERSION ----- In 1979 I defined a new version of COMAL to be implemented on a microcomputer. This version - COMAL-80 - was further improved by a working group with members from The Technical University of Copenhagen, the University of Roskilde, and representatives of several Danish manufactures of microcomputers. COMAL-80 includes such facilities as parameter passing (both value and reference), local variables, and recursiveness. It has been implemented on the Commodore CBM-8032 and 4032 microcomputers, the Commodore 64, and two Danish made ones, the RC700 and the ICL COMET (I wonder if ICL, England, knows that they have in fact a very good COMAL running in Denmark?).

THIRD AND SAD CHAPTER ----- Happy ending, is it not? Well, not quite. I'm sorry that I have to write patient reader a third and sad chapter about "basic". A few days ago I met the word again in an important and fatal context. I got a paper from England titled "The BBC Microcomputer. Outlined specifications of the BASIC language interpretter". The paper does not reveal its author(s), and so far that is the only trace of good taste I can find in it. It is most laudable that this glorious institution, BBC, intends to teach the Englishmen about computers and programming. And of course those responsible for such a worthy enterprise know that it must be carefully prepared and that both hardware and software goes with it. But are they fully aware of the crucial importance of the programming language that may be used by hundreds of thousands of viewers. Not only for programming purposes, but to a still higher degree for COMMUNICATION OF IDEAS.

CONSEQUENCES ----- Programs for computers are of increasing importance and more are being written every day. These programs are meant to be used for something. They are going to DO something which in most cases immediately influences peoples welfare in the broadest sense. Economy, health, culture. It has been said that the computer is a tool to extend the human brain in much the same way as the steam and combustion engine radically extended the potential of the human body. I think it is more precise to say that the computer extends our linguistic potential. Man has always been able to use the language to trigger off movements and changes in his environment. But it is new that we are now able to leave the traces of our language in tools which then, maybe long after the words have fallen and could be far away from the place where they were spoken, translate the message into action. The consequence may be new knowledge, new potentials, new wealth, but also disasters that may leave a long trace in our history. At the beginning the word was there...

FUTURE JOBS ----- If one considers what language means to man in general, in our intercourse with each other, in our understanding of the world and ourselves, it is almost incredible to watch the improvidence with which we have designed and used programming languages. As the number of problems grow, it will be of growing importance that we can READ each others programs. The message from one person to another will also in this field be of major importance over the communication with the computer. In his book THE GLOBAL CHALLENGE the french author Servan-Schreiber says, in the chapter about the new information technology, "No industrial country will be able to survive the global revolution, if it does not create FUTURE JOBS with this revolution as a starting point. The necessary readjustment consists in leaving the outdated pure commercial competition, the aim of which is the capture of markets, and which has lasted 30 years and is played out. Instead we must institute a new competition that is based upon the EDUCATION OF MAN, the training of brains, of the ability to create..."

WHERE BBC COMES IN ----- And this is where BBC and its computer course comes in. In England you have a great tradition for good programming. Names like C.A.R. Hoare and Elliot computers come to mind. And the rest of Europe. Where were languages like Algol and Pascal invented? This is where we are much better than the Americans and the Japanese. Try to compare the miseries of Fortran and Cobol, not to speak of PL/1, the greatest scandal since the tower of Babel - with the elegance and power of Pascal, and you know what I mean. This is where WE can compete. And this is the field that matters in the futue!

But instead of seeing this, BBC unconsciously wants you to ape after the Americans, in a field where they are definitely bad, and where you could be very good. But we have to use BASIC, the cowards yell. Anybody uses BASIC. A hell of a lot of programs have been written in BASIC. Now, take it easy. That a lot of programs have been written in BASIC doesn't matter. They are very few compared with the immense amount that are going to be written in the future. And BASIC is not used by anybody. A lot of people are using BASIC, admittedly, but they are not the most important ones. And they are a minority compared to all those who will use computers in the future. In Denmark we have learned that people will turn away from BASIC if only they get something better.

FOR AND AGAINST ----- But BASIC is the only language that can be put into small and cheap

microcomputers, is another argument. Not true. The early versions of COMAL only took up 12% more storage that the BASIC interpreter, and we had not thrown away the GOTO, GOSUB, RETURN, ON GOTO, which we could easily have done without, and which in fact have not been in use since 1976. On page four of the heroic paper about BBC-BASIC it says - about the extensions to a BASIC which is neither flesh nor good red herring - "These extensions should be avoided in simple programs that are intended to be used on a variety of machines". Among the extensions is alot of the same hopeless jumble that characterizes the rest of the design, but one or two important ideas are hid in between, namely those of LOOPS and NAMED SUBROUTINES.

MEET THE GLOBAL CHALLENGE ----- And that kind of extensions should NOT be avoided in simple programs. On the contrary. Beginners should be urged to use such facilities, and at the same time be kept away from using the GOTO which is a very advanced tool only to be used by well trained and very good programmers. The whole concept has to be carefuly redesigned, if BBC is not going to spread a mortal disease among the next generation of British programmers and users of computers. Meet the global challenge, where you are able. Where did British motorcycles go? And how are British steel, ships and cars these days? Is programming and control of computer systems to go the same way! Don't let it happen. BBC, think again!

Yesterday Today credits:

Len Lindsay: Editor
Frank Hejndorf: Captain COMAL Circleman art
Wayne Schmidt: Calvin the COMAL Turtle art
Bill Nissley: Draw Poker Screen Image art
Angus Quinn: Inside Cover Circle People art
G Raymond Eddy: Video Cartoon art

Articles and Programs by:
Richard Bain
Jack Baldrige
Borge Christensen
Glen Colbert
Captain Buzz COMAL
Garrett Hughes
Debra Ruth Junk
Dick Klingens
Len Lindsay
Kevin Quiggle
Robert Ross
David Stidolph
Colin Thompson

N58----- COLOR COMBINATIONS -----
The Commodore 64 computer has some color combinations that yield unreadable text on some TV's. So we would like to know what color combinations are unreadable on YOUR system, and which are BEST. Photocopy the chart below. Then RUN the color program. Fill in the chart with your judgement of the color combination readability on YOUR SCREEN (be it black & white, green, amber, or color). Use the following rating: 3=Excellent, 2=Good, 1=OK, 0=Poor. Send the chart to COMAL TODAY, 5501 Groveland Ter, Madison, WI 53716. We will compile all charts submitted and come up with a composite chart. Do it today!

```
BACK'COLOR:=0 // hit any key for next set
REPEAT // hit STOP key to STOP
 BACK'COLOR:=(BACK'COLOR+1) MOD 16
 BACKGROUND BACK'COLOR
 PRINT CHR$(147) // CLEAR SCREEN
 FOR PENS:=0 TO 15 DO
  PENCOLOR PENS
  PRINT "PENCOLOR";PENS;"BACKGROUND";BACK'COLOR
 ENDFOR
 WHILE KEY$=CHR$(0) DO NULL // WAIT FOR KEY
UNTIL TRUE=FALSE // forever
```

3=Excellent 2=Good 1=OK 0=Poor

PENCOLOR

BACKGROUND	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	BACKGROUND
0	0																0
1		0															1
2			0														2
3				0													3
4					0												4
5						0											5
6							0										6
7								0									7
8									0								8
9										0							9
10											0						10
11												0					11
12													0				12
13														0			13
14															0		14
15																0	15

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

PENCOLOR

3=Excellent 2=Good 1=OK 0=Poor

My TV / Monitor is a:

PRINT FILE and WRITE FILE - the Difference

A PRINT FILE statement is just like a PRINT statement, except the output goes to the file instead of the screen or printer. For example: If NAME$="JOHN" then

```
PRINT NAME$," IS HERE"
```

would print on the screen: JOHN IS HERE.

```
PRINT NAME$,NAME$,NAME$
```

would print on the screen: JOHNJOHNJOHN. The comma means tab to the next zone, which by default is 0 (no extra spaces). How about this:

```
DIM NAME$ OF 40
OPEN FILE 2,"TEST-PRINT",WRITE
NAME$="JOHN"
PRINT FILE 2: NAME$," IS HERE"
PRINT FILE 2: NAME$,NAME$,NAME$
CLOSE FILE 2
```

```
RUN
```

Now, instead of printing to the screen, the results of the PRINT go to the disk file. The example created a disk file with two records. These records are retrieved like this:

```
DIM TEXT$ OF 80
OPEN FILE 3,"TEST-PRINT",READ
WHILE NOT EOF(3) DO
 INPUT FILE 3: TEXT$
 PRINT TEXT$
ENDWHILE
CLOSE FILE 3
```

```
RUN
JOHN IS HERE
JOHNJOHNJOHN
```

The second program opened the same file and read the two records, printing them on the screen. Each PRINT FILE statement stores its data as a long record with a CHR$(13) delimiter at the end. Or you can put a CHR$(13) in with the data to create your own record end delimiter. The file is in ASCII, just like Commodore BASIC. You can also use INPUT FILE statements to read a BASIC text data file or to read a COMAL program that was LISTed to disk (ie, LIST "PROGRAM.L").

WRITE FILE and READ FILE also store and retrieve data to and from disk. However, they are record oriented, and the file is a binary file (more efficient and compact).

```
DIM NAME$ OF 40, TEST$ OF 20
OPEN FILE 2,"TEST-WRITE",WRITE
NAME$="JOHN"
TEST$="IS HERE"
WRITE FILE 2: NAME$,TEMP$
WRITE FILE 2: NAME$,NAME$,NAME$
CLOSE FILE 2
```

Note that only variables or array values can be used with a WRITE FILE statement (in version 0.14). Thus

```
WRITE FILE 2: NAME$," IS HERE"
```

is not possible (version 2.00 cartridge will allow it). So, we assigned "IS HERE" to a variable. A comma is used to separate records in a WRITE or READ statement.

```
DIM TEXT$ OF 80
OPEN FILE 3,"TEST",READ
WHILE NOT EOF(3) DO
 READ FILE 3: TEXT$
 PRINT TEXT$
ENDWHILE
CLOSE FILE 3
```

```
RUN
JOHN
IS HERE
JOHN
JOHN
JOHN
```

The results are different. The results are just what you need when writing and reading data records. That is why it is highly recommended that you use WRITE FILE and READ FILE instead of PRINT FILE and INPUT FILE.

Thus, if you would like to save the values of three variables to disk

```
WRITE FILE 3: X,Y,Z
```

is what to use.

```
PRINT FILE 3: X,Y,Z
```

would not work.

```
PRINT FILE 3: X,CHR$(13),Y,CHR$(13),Z
```

would work, but looks like the mess many are used to from BASIC programming.

L21 --- ...it seems that COMAL is exactly what I have been waiting for, even allowing for learning a new language. W.C. Delaware

THE PROBLEM OF INERTIA
BY COLIN F THOMPSON

Fully one third of the members of C64 WEST have now had some experience with COMAL. This is pretty astounding, considering the COMAL SIG (Special Interest Group) has only been operating for 2 months. As SIG leader, I receive one or two telephone calls a day from COMALites in the area, seeking information and solutions to programming questions. Why has COMAL caught on? **WHAT'S THE FUSS?**

As I see it, the answer is psychological. Many of us bought a home computer just to see if the experience 'agreed' with us. We wanted to know if the computer could be conquered. For most, the answer was a qualified Yes. We found that some of the aspects of home computing could be mastered and enjoyed. By using commercial software, we found the computer could lighten our workload, thus leaving us more hours in the day to enjoy ourselves. Others found the Joy of Telecomputing and now concentrate soley on Modem and BBS activities. Other areas of interest include Game playing and Education. Computer clubs thrive on this diversity of interests. The Club brings together people skilled in one or two computing activities and lets them meet others with skills they might like to learn. This social structure, with its foundation of interpersonal relationships is the same one that our civilisation is based upon.

BUT WHY COMAL?

Programming is the one area of computing interest I failed to mention. Wouldn't it be fun to make the computer follow YOUR instructions instead of the prepackaged instructions found in commercial or public domain software? Apparently the answer, for many, is a resounding YES. Since most of us come to the C64 with absolutely no computing skills, we are expected to learn the programming language that comes with the computer: Commodore BASIC Version 2. This fundamental assumption can be the downfall to many who might like to learn a programming language, but find their first brush with BASIC to be a 'brush off'. BASIC is not easy to learn.

BASIC is ancient (by modern standards). It was first developed about 25 years ago to allow neophyte programmers to learn how to write programs witout having to know exactly how the computer worked. Before that time, the only programmers were hardware engineers and scientists. So, at that time, BASIC was a boon; a godsend. Times have changed and programming languages have changed also. BASIC is now considered the absolute worst language to inflict on a novice programmer.

If it's so bad, why is it so popular?

INERITIA.

No manufacturer of computers wants to rock the boat by including a 'non-standard' language in their computer. As long as there have been Home computers, BASIC has been the only language to be built in. Most manufactureres actually beat their chests and exclaim loudly that their BASIC is better than the other guy's.

Why do computer makers resist change while other technological industries quickly embrace the latest inovations? That's a tough one. Imagine the auto industry trying to sell you a car without steel belted radials. The old bias ply tire simply couldn't compete with the radial. Unfortunately, BASIC is the bias ply tire of the computer industry. It simply pales by comparison to COMAL, yet Commodore and the others still cling to it like a security blanket. That is a sad commentary on the attitudes displayed by the key people in our industry. They talk NEW but sell OLD.

Not every computer owner needs to know how to write programs. Those days are long gone. It's now your option to learn a language. That's the way it should be. If the idea of learning to write programs attracts you, don't make the mistake of assuming that BASIC is the only language available. At this point in the evolution of programming languages, COMAL has the most to offer the beginner.

TAKE THE FIRST STEP

All the tools you need to begin your jouney into COMAL are at hand. The language, books, newsletters, programs, and user groups are at your disposal, but YOU must take the first step:

LOAD "BOOT C64 COMAL",8 and say goodbye to BASIC.

ERROR MESSAGES BUILT INTO COMAL
BY GLEN COLBERT

One of the most common complaints about COMAL version 0.14 for the Commodore 64 is waiting for it to retrieve error messages from the disk drive when a mistake is made. Of course, there is always the option of deselecting the disk error messages, but the resulting **ERROR ##** is a poor substitute for the truly nice warnings given by the COMAL system. The RAM ERRORS program provides a solution to this common complaint.

The RAM ERRORS program places the error messages into memory and re-writes the COMAL error routine to find the error messages in their new home. Additionally, each time an error is entered a bell is rung. The message is then promptly printed to the screen. Errors are no longer as exasperating as they once were, and no disk access is unnecessary.

One of the problems that has to be overcome in placing the error messages in RAM is finding a place to store them. My solution was to use some of the space reserved for sprite images. RAM ERRORS stores the contents of the COMALERRORS file starting at memory location 50112. This area (between 49152 and 52735) is used to store the sprite images. Unfortunately, placing the error messages here restricts the use of the sprite images, **but does NOT prohibit it.** Because there are different versions of the COMALERRORS (English, Spanish etc.) file, it was important to keep the ending area for the error messages variable. With the standard COMALERRORS file, sprite images 15 to 35 are strictly off limits. DO NOT USE THEM WITH RAM ERROR MESSAGES!! (You may still use SPRITES 0 thru 14).

One of the nice things about COMAL is that it permits the user to define the error messages. Spanish, German, French and many other versions are available [Ed Note: we hope to make these files available on TODAY DISK #5. If you are using a special ERROR FILE send it in]. This program was written to make life easier for those of you who use a non-standard error message file. Rather than force its own error messages on you, this program will read your error message file and enter it into the computer. Be sure that you have a disk in the drive with your COMALERRORS file on it when you run this program.

While entering this program, be extreeeeemly careful to enter the data correctly. Be sure that you **SAVE THE PROGRAM BEFORE RUNNING IT.** Any errors in the program after it is run will necessitate turning off the computer and re-booting COMAL. This program is written for C64 COMAL Version 0.14 only!!!! For your convenience, this program is on TODAY DISK #4.

Once a good copy of the program has been entered and saved, RUN it and begin a bit of experimentation. Try POKEing different values into memory location 6475 (e.g. POKE 6475,100). Then enter an error intentionally. This location gives the value for the error bell. By POKing different values into it, the sound can be changed. If you find a value that is more pleasing, it can be entered into the RAM ERRORS program (the third data line). This way, it is the sound used each time the program is run.

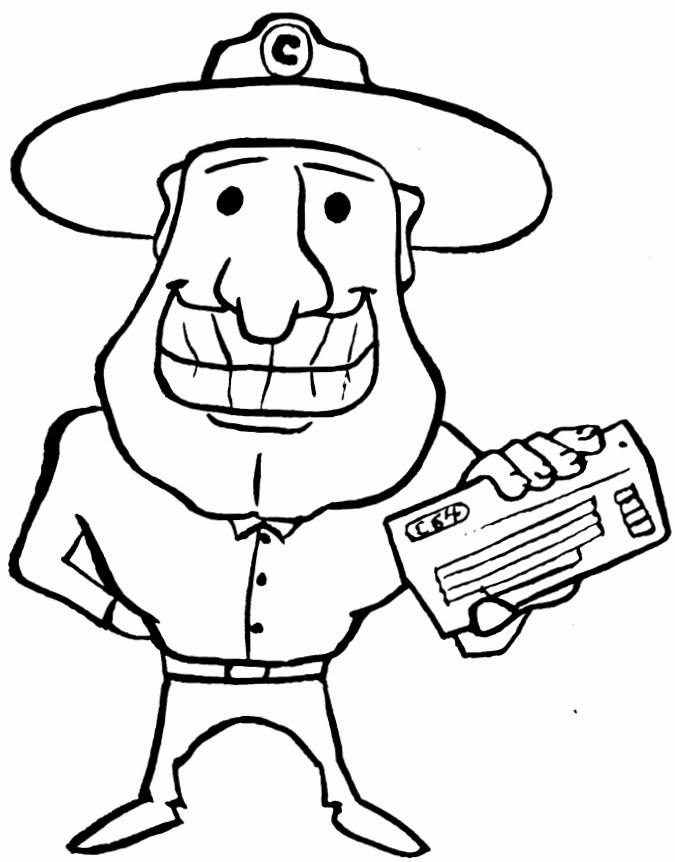
If you don't care for whistles and bells, the error message sound can be shut off. The SETMSG command is used to select this option. SETMSG- turns the bell off and SETMSG+ activates it (you should turn it off right away if that is your choice).

I think that you will be very pleased at this modification to COMAL. While it is still far short of the really nice error message routine used in version 2.0 cartridge, it is a significant improvement over accessing error messages from the disk.

[Ed Note: This is very useful addition to COMAL 0.14. It does **NOT** use up any of your free RAM workspace. It's only drawback is that you can only have 15 different sprite images (0-14). However, most programs do not use sprites at all, and those that do usually will work fine with only 15 images.

We assume that most people will want this NEW feature all the time, so the HI program on TODAY DISK #4 includes a modified version. When asked if you want error messages, reply **Y** and get RAM ERRORS with sound on. Reply **N** and you are in plain COMAL 0.14.]

```
//delete "0:ram'errors"
// by glen colbert
//save   "0:ram'errors"
disk'get'init
print chr$(147),chr$(14)
print "          ERROR MESSAGES IN RAM"
print "         (c) Glen Colbert 9/2/84"
print "  This program loads the error"
print "message file into ram starting at"
print "50112 and ending at the end of"
print "the file length plus 50."
print "  This is the same area of memory"
print "used by SPRITE IMAGES 15 - 56."
print "Using these sprite images will"
print "cause COMAL to CRASH AND BURN!!"
print "Insert a disk with the file"
print "       COMALERRORS"
print "then press return."
while key$<>chr$(13) do null
print "Reading file."
open file 2,"comalerrors",read
position:=50112
file'end:=0
while not file'end do
 a:=disk'get(2,file'end)
 poke position,a
 position:+1
endwhile
close file 2
print "Changing error routine."
endit:=position+50
while endit<>position do
 poke position,255
 position:+1
endwhile
position:=6462
while not eod do
 read a
 poke position,a
 position:+1
endwhile
end
// disk'get routines follow
//
func disk'get(file'num,ref file'end) cl
 osed //continuation line from above
 poke 2026,file'num
 sys 2025
 file'end:=peek(144)
 return peek(2024) //value of character
endfunc disk'get
//
proc disk'get'init closed
 for loc#:=2024 to 2039 do
  read v
  poke loc#,v
 endfor loc#
 data 0,162,0,32,198,255,32,207
 data 255,141,232,7,32,204,255,96
endproc disk'get'init
//
data 134,45,173,216,16,240,36,169
data 15,141,24,212,169
data 74 //change for new sound
data 141,5,212,141,0,212,141,1,212
data 169,17,141,4,212,169
data 0,168,170,202,208,253,136
data 208,250,169,16,141,4,212,32
data 2,107,169,192,141,170,25,169
data 195,141,171,25,32,167,25
data 201,255,240,41,197,45,240
data 17,32,167,25,24,105
data 1,133,43,32,167,25,198,43
data 208,249,240,228,32,167,25
data 133,43,32,167,25,133,25,32
data 167,25,32,210,255,198,43
data 208,246,96,160,0,185,192
data 195,238,170,25,208,3
data 238,171,25,96,0
```



Book Club Feature

You know you're getting somewhere when you get to be a FEATURE book for the Byte Book Club. COMAL finally got there. The midsummer 1984 list had the COMAL HANDBOOK as the FEATURED ALTERNATE book.

STRINGS IN COMAL

by Len Lindsay

Several people have now written asking about the missing MID$, LEFT$ and RIGHT$ in COMAL. Actually, these are not missing, they are just implemented differently (better after you understand the COMAL way). Appendix B of the COMAL HANDBOOK explains COMAL string handling. This article will only cover some important aspects.

What MID$, LEFT$ and RIGHT$ are actually doing is providing your program with a SUBSTRING. In COMAL, you chose the substring by naming the string followed by the characters needed specified inside parentheses:

```
A$(<start character> : <end character>)
```

If DATE$="12/25/84" we can find the month, day, and year by using substrings:

```
PRINT date$(1:2) // print month
PRINT date$(4:5) // print day
PRINT date$(7:8) // print year
```

In Microsoft (Commodore) BASIC this would look more contrived:

```
PRINT MID$(D$,1,2):REM PRINT MONTH
PRINT MID$(D$,4,5):REM PRINT DAY
PRINT MID$(D$,7,8):REM PRINT YEAR
```

Of course you can use variables to chose the start and end character positions. Both start and end positions are optional. For example, to specify just the third character in the string:

```
PRINT date$(3)
```

If you want the first five characters, simply use 1 as the start:

```
PRINT date$(1:5)
```

But the best part is that you can change any part of a string without affecting the rest of it (not possible in BASIC). For instance to change the third character from "/" into "x" we would simply type:

```
date$(3):="x"
```

Now, date$ is equal to "12x25/84".

This same string handling applies to string arrays too. Just specify the substring points right after the array index:

```
PRINT NAME$(5)(1:6)
```

This would print the first 6 characters of the 5 element of NAME$ array. It is all consistent, so you also can change any part of any element of a string array.

I'll conclude with one subtle point. What if you specify a range of characters to be changed, but don't assign enough characters to it? Well, COMAL will just fil in spaces to get to the end character. For example:

```
A$:="ABCDEFG"
A$(3:5):="X"
PRINT A$
ABCX  G
```

Use of this feature may not be apparent, but it does come in handy. For instance, you pad spaces on to the end of a string to make it exactly 20 characters long easily now:

```
NAME$(1:20):=NAME$
```

Did you catch that? We told COMAL to assign characters 1 thru 20 of NAME$ to be equal to NAME$. Now if NAME$ starts out as "BOB", COMAL will add spaces to it until the 20th character. It also is easy to blank out a string variable now by turning it into all spaces rather than just a null string:

```
NAME$(1:20):=""
```

For more information on strings, see the COMAL HANDBOOK appendix B.

BUG IN COMAL TODAY #2 PAGE 29

There was a bug in the program listed on page 29. Procedure LP'CONDENSE needs a quick change:

```
OLD: OPEN FILE 13,"",UNIT 4,8,WRITE
NEW: OPEN FILE 13,"",UNIT 4,13,WRITE
```

CHEER UP - IT COULD BE WORSE

or

TALES OF THE WEST COAST COMMODORE SHOW

by Colin Thompson

[ED NOTE: You will have to read this several times to catch all the fun Colin has included. This is Colin at his finest. Please read it in that light]

It sounded too good.

The man on the phone had a distinct English accent and was using it well. "Malcolm Lowe," he said, "and I'd like to invite you to speak about COMAL at the West Coast Commodore Convention in February".

Following the standard offer of airfare and hotel expenses, he asked if I could make it. "Well, let's see," I mumbled, as I thumbed through my appointment book. Ah, here's February, and - ooops, I'm due to speak in Calgary on that weekend. Calgary!? What lunatic agreed to go to Calgary in the winter?

"This lunatic," I said. "What's this about a lunatic?" Malcolm politely asked. "Sorry, just reading out loud, Malcolm. I seem to be scheduled for Canada then, but let me get back to you in a couple days. Maybe something will change."

"Something will definately change," I swore under my breath. Bye, Malcolm. Quick, what's the number of the Calgary group?

Two months and several phone calls later, I meet Len Lindsay at the San Francisco airport. I've decided that this was a good opportunity to preach COMAL to my favorite kind of group, a captive audience, so I've invited Len to lend some authority to the event. This is Len's first visit to San Fransisco, so I play tour guide during the Limo ride to the hotel. He's amazed at the sight of a strange city, jammed with concrete ribbons of freeways. I'm amazed at the weather- it's not raining.

Following the ritual of tips, check-in, credit cards, tips, bellmen, baggage, tips, long corridors and tips, we set out in search of some friends, who, we have on good authority, are staying in the same hotel. Our first target is Jim Butterfield, who hasn't yet checked in. Next is the Computer Curmudgeon, Mindy Skelton, who has already taken up residence, but cannot be found. We eat.

While enjoying our lunch of red snapper that looks and tastes just like sole, we remember that Randy Chase was going to attend. The front desk (the computer, really) informs us it has no record of a Mr. R. Chase, of Oregon. Strike three.

Idle curiosity moves us to the mezzanine, where the show is being set up. Chaos. Wires, lights, boxes, frantic people. We get our badges (yes, we don't need any stinkin' badges, but we pin them on anyway). Now armed with a visual "right to exist", we enter the center of the melee, the combat zone, the main floor. Any trade show, the day before it opens, looks just like this. Kinda like a zoo, populated by a commitee. We walk.

Steve of PP+S tells me he has a great new product. I never get to see it. Tom Lynch, President of the Valley Group invites me to his booth. He has written a COMAL Cartridge program that is running on his 1702, shouting the advantages of belonging to THE user group of north LA. I liked the program.

We walk in search of the PlayNET booth. Len assures me he can recognize Mindy - he met her last year. Cathy "Miss PlayNET", says Mindy hasn't been there for a while, so we turn to walk farther. Wham! Bang! I'm surrounded by arms. Much hugging. I nearly fall down. Mindy helps me to my feet. Len, ever the gentleman, makes a tardy and now unnecessary introduction. "Let's eat. I want to tell you both about PlayNET," commands the Curmudgeon. We head for the coffee shop, following in the wake of a person who has enough excess energy to sell to Three Mile Island. Seated in the now familiar setting of the coffee shop, Mindy tells us that Randy can't make it. Something about having a magazine to get out on time. She begins an account of her new existance as a PlayNET Junkie as Jim B walks in. Len excuses himself to confer with The Man.

Mindy continues, undaunted. I listen. PlayNET, it seems, (I have not seen it at this point), was created by God, Herself,

and given to the World Of Commodore, (which She also created). Mindy prefers PlayNET to sex or food. I begin to wonder about Mindy. She says we MUST go to the PlayNET booth after the phone lines are installed and she will give us both an electronic tour of it's chambers of delight. "I can't wait," I respond.

Len returns, and we repair to our respective rooms to "dress" for the evening's entertainment. Commodore is holding a reception for user group functionaries, followed by a gala given by the WCCA organizers for the vendors and speakers. I surely wouldn't want to miss either, so I suit up in my Sunday finest, replete with tie. We three meet in the lobby. They don't seem to recognize me. I introduce myself. They are clad in the latest version of "California Casual"- jeans, tee shirt and running shoes. I'm wearing a tie. I feel out of place, they let me come along anyway.

I know we are close to our target destination, a suite on the 7th floor, when I see Jim B holding forth in the crowded corridor. We squeeze by and enter the room. It's packed with user group luminaries and Commodore officials. The officials shout about their newfound "support" of user groups. The luminaries ask detailed questions. The officials mumble into their drinks.

I look at the sign-in register and discover, to my delight, that some folks that I correspond with are here in person. I start looking at nametags. Most of the nametag owners stare back. The next hour is spent in delightful conversation with John Zacharias of Sacramento, Craig Borden of Tulsa, Jane Campbell of San Diego, Shelly Roberts of Nu Yawk, and Rich Tsukiji of Oregon. Shelly introduces me to Wes James, a commercial programmer and NYCIG's Guru. Wes asks what's a COMAL. Wrong question, Wes. I talk, he listens.

Wes finally dismisses me with an "I prefer compiled BASIC" and we adjourn to the ballroom, downstairs. The Commodore Contingent is led by Pete Baczor and Jim Gracely, earnestly explaining why they think the new model 128 will become the sweetheart of the marketplace. They haven't heard of an "Amiga". They DO know all about the new lap computer, the LCD. Whenever asked a technical question about the LCD, they wrinkle their brows, think a minute, then point, in tandem, to a young man near the punch bowl. "He knows. Ask him. He designed it," they whisper. Pete plays Oliver Hardy and Jim does Stan Laurel. I enjoyed the act.

I don't much like parties. I tend to find a corner and just hang out, wishing I owned a good Lap Computer to pass the time. The corner I picked was keeping company with Guy Wright, Mr. RUN Magazine. He likes corners also. We talked for nearly an hour, while observing the high priesthood of The Church of Commodore give blessings on their adoring flock.

All good things must end (Book of the Kernal 3:12), and thus it came to be; I was "rescued" from my corner and made to join 11 other revellers on a Mission From God - it's time to eat. Dick Immers knows a great Tibetian Barbeque that's "just around the corner". We set out on foot, following him. He leads us on a tour of San Francisco's alternate lifestyle district. Len stares. I cringe. We all hurry, but this doesn't seem to be Little Tibet. We pass the Hard Rock Cafe. The line goes around the corner, so we settle on a tacky little Italian place that features a 99 year old man playing the theme from the Godfather on a 99 year old piano. We are the only patrons. Maybe the rest of the city knows something about this place that we don't.

Dick tells how much fun he had writing his book on the 1541. Wes announces he has completed the first true CAD system for the 64 called CAD GEM. It's being marketed by Commodore and will be out in a few weeks. It's a real 'wire frame', three axis, hidden line design system with a screen size of 2 megabytes - all for only $90. I write a check on the spot and forgive him for his comment about compiled BASIC.

Next stop on our whirlwind tour is the bar in the hotel. There our party fragments and I wobble to my room, hoping to catch a few hours sleep to prepare myself for the rigors of opening day.

What rigors? I have nothing to do. Our lecture is on the following day, Sunday,

so I rise with the roosters and wobble to the coffee shop. Len has arranged to share booth space with Loadstar and I decide to sit in on some of the morning presentations.

I've attended dozens of trade shows, but have never actually listened to a lecture. Amazing. Well, my initiation began at 10 AM when Laurel and Hardy gave their presentation. They have a new program designed to support Commodore User Groups. Apparently Commodore just found out about User Groups. The main thrust of their new program is an eight page newsletter - called Input/Output. The issue I got seemed to be slanted toward Output. Many of the pages were given over to the important task of revealing the software available for those dynamite computers, the Plus Four and the Commodore 16. Great stuff, and Commodore will actually send this newsletter, free of charge to any user group that applies for "qualification" and gains "approval". I fill out the application form with hands trembling in anticipation.

Stan tells us about Commodore's long term plans. (In the long term, we all die). Olly tells us about the new C128. It has three "configurations", not to be confused with "modes". Semantics are important to someone in Upper "Management". We are introduced to the first configuration - called the "Sixty-four mode-er-configuration". It is completely compatable with all existing C64 software and hardware. Next, the 128 mode-etc. This is a real gem, we hear. It has a lot of memory - 122K free, and uses the 1571 Disk Drive. The 1571 is a Real Neat Thing. It can tell what kind of a diskette is in the drive and alter it's bus transfer rate to match the microprocessor needed to RUN the program.

Did you get that?

Maybe I should explain the third mode-configuration. It's called CP/M and Commodore is really pushing it. They have convinced Thorne-EMI to rewrite the Perfect series CP/M programs to run on the 128 in CP/M mode. CP/M is what's called 'disk intensive' - it uses the disk drive all the time, so the bus transfer rate had better be fast or it will take all day to write a memo on WordStar. We all know that Commodore has the slowest disk drives in this universe and probably the next. In order to use CP/M, the 1541 had to be coaxed into HyperDrive. That's where the 1571 comes in. It has a three speed automatic transmission.

In the C64 mode it behaves just like a 1541 - 300 Characters Per Second transfer rate. When the 128 mode is selected, the drive steps up to 1500 CPS. That's a step in the right direction, about 8 years too late. Finally, the CP/M mode needs some REAL speed and gets it - 3500 CPS. This may prove to be a workable combination. Time will tell.

Speaking of time, I wonder how many hours will be spent trying to jimmy the Operating System into letting the C64 mode use the 3500 CPS disk rate? Carl Sagen's "Billions and Billions" sounds like a low estimate.

The LCD looks like a good machine. It has a big, fold up LCD screen display. It does not have Easyscript built into ROM. Too bad. I would have bought the cute lil' bugger if I could write ES articles on the plane, or some other inconvienent place. They showed off a 3 1/2 inch Sony drive that plugged into the LCD. That drive intrigued me. I wanted one or two for my C64. If they actually release it, I might be able to use it on the 64. At least that's what the young man who designed the system told us. You could plug in a 1541 also, but not a monitor or TV set. The LCD's built in software looked similar to the NEC's built in software. After hearing Commodore talk about all the good things the LCD will do, I think I may be able to write text files on it, then download them to my 1541 at home. Then I could bring the files in EasyScript. That's almost a workable plan. Nice going, Commodore.

The last 15 minutes of the presentation were spent answering questions from the audience. Johnny Carson never had to contend with an audience this hostile. It seemed every other question was "What about the Amiga?". And of course, every other answer was "I don't know anything about an Amiga, or a Lorraine, or any other 68000 based microprocessor that the trade press has prematurely announced". End of story? Not hardly.

All throughout the presentation, Commodore's young, unnamed genius, tried to make the C128 do something - anything. It, of course, would not budge. Murphy strikes. When the show concluded, I wandered up the the stage to get my hands on a keyboard. I remember the dismay I felt when I first touched a Plus Four. I'm sorry to report that Commodore gave the keyboard contract to the same firm that ruined the Plus Four. Too bad. I'll just wait for the phantom Amiga. Jim B swears it will be out by next February, or April at the latest.

I don't recall much of what went on the rest of that day. Just a blur of people without names, another attempt at finding Little Tibet, and then Bed.

Bright And Early Sunday Morning. Len and I are scheduled to speak at 11 AM, an hour after the show opens it's doors. Once again, I'm sporting my tie, etc. Over breakfast we plan how to divide our talk. I'll open with a five minute overview, then hand it over to Len. He does 15 minutes on the State OF COMAL, then I do 20 minutes of technical discussion. That leaves Len 5 minutes to describe the function of his COMAL Users Group, USA. With our plans firmly set in mind, we walk to the conference room.

Eleven AM. The computer is set up and all the hardware is working. On the overhead projector, the COMAL Cartridge is showing off. The room will hold 400. 40 of the chairs are filled and we begin. I start to speak. The room reverberates with sound of empty chairs. I continue, regardless. Thirty minutes later, I conclude my five minute intro, and hand it over to Len. He talks until Malcolm politely unplugs the mike. Our time is up. The crowd surges to the stage and we begin to answer all the questions that were considered 'too stupid' to ask in front of everyone. We politely answer them until Malcolm's security agents escort us from the hall. Alas, 45 minutes is simply not enough time.

I approach the President of the WCCA and politely inform him I would consider doing this again, only if I'm given four hours, on a Saturday afternoon to make my presentation. He agrees. Len and I pack up and fly to Santa Monica. Now he sees Real freeways. He is amazed. As the cab nears my apartment, I realize that we have not seen a drop of precipitation in three days. The temperature is 72 degrees, and this is February. I wonder what it's like today in Calgary.

REVIEW - GRAPHIC PRIMER

Book / disk set - (special now $14.95)
reviewed by David Stidolph

Three different textbooks have been published to help people learn COMAL. However, none of these textbooks teach anything about COMAL's graphics or sprites. Now, thanks to Mindy Skelton, COMALites can learn about the graphics abilities of their computer. The book, over 80 pages in length, is a good introduction to drawing graphics and sprite control.

Beginning with a short introduction to programming and procedures, it moves quickly to "turtle" graphics (turtle graphics is the system of drawing that COMAL uses). The approach in teaching graphics is usually done while the student actually tries the commands as they read about them, and this book follows that format. Each command is described in the order that you should learn them, so there is little need to "jump" around the book.

Sprite control is one of the hardest graphics features to learn, but Mindy's book comes to the rescue with step-by-step instructions on what a sprite is and how to create and move it. Not one or two, but three ways of creating sprites is shown.

Many sample programs, and useful procedures and functions are presented, each with explanations on what they do. Finally, the book has a good index and a complete glossary of ALL graphics and sprite commands, including the GETCOLOR command, which was not known to be a valid function until this book was published.

Since the book comes with a matching disk that contains all the programs from the book typed in, very little typing is needed to try the examples. The disk has a menu of the sample programs, so you just LOAD and RUN that program, and you can select any of the examples to RUN.

GRAPHIC PRIMER is a good tutorial for COMAL 0.14 graphics. I highly recomend it.

Illiterates Unite!

"What's a turtle?"
"It's a sprite."
"What's a sprite?"
"It's a little creature that runs around the screen."
"What's a screen?"

Yes,there are a few people left in the U.S.A. (or one for sure) who know nothing about computers. Before becoming a staff member of COMAL TODAY, I thought RUN was a magazine for joggers and BYTE a journal for orthodontists.

Fortunately, the optimistic editor of COMAL TODAY sees this ignorance as a plus. There may be a few readers who are new hands in the field, and documenting my journey through the maze of computer-ese may answer the questions of others.

So, what exactly is a turtle? It's a drawing tool for the computer. The turtle is used the same way a pencil is used on paper, the major difference being no one ever called a pencil a snake or a worm. A pencil is a pencil --- we don't come up with different cute names for it in hopes of purposely confusing people.

The turtle shows up on the screen as a small triangle or arrow. It has a pen attached to it and as you move the turtle around, the pen writes. The turtle is quite docile and responds well to the following commands:

TURTLESIZE: the turtle can be made bigger or smaller in an instant, at the whim of the person at the keyboard. (I think this command should be re-named "turtle-on-a-diet.")

HIDETURTLE: if you're mad at your turtle and don't want to look at him/her, you can command HIDETURTLE and the turtle will disappear from the screen. However, the turtle will continue to draw for you while in this state of exile. He really is one heck of a nice guy.

SHOWTURTLE: When you are ready to readmit the turtle to your kingdom, punch in SHOWTURTLE, and a turtle will reappear on your screen. Don't assume this is the same turtle you had before: he may have tired of your little games and sent in a replacement.

PENUP and PENDOWN: PENDOWN is easy to understand. It simply means the pen is touching the paper, so to speak, and you can draw. But if a turtle's only job is to draw, why do we need a PENUP order? Len came to the rescue and explained it this way: Suppose you drew a house and then wanted to add a sun to the sky. Would you want a line to be drawn between the house and the sun? No. So you need to pick up the pen as you move the turtle. This made good sense, even to me.

Fellow novices, take heart. I learned all of this merely by watching Len and his turtle at play. I didn't touch the computer even once. Actually, I'm not anxious to start fooling around with these suspicious-looking machines. Does that glazed look in the eyes of the computer person appear instantaneously the first time one touches the keyboard? But I will set these worries aside, as duty calls and the adventure begins. (Next month: The Art and Science of Loading a Disk).

A bit of comic relief was provided to all attending the COMAL Standards Meeting in Sweden by Angus, a representative from Scotland. He drew a caricature of each person attending the meeting. (see inside front cover)

The Truth About Computers

by Debra Ruth Junk

Two erroneous beliefs concerning computers have taken root in American society. One holds that computers are logical, scientific and programmer-controlled. The other declares that computers are essentially stupid and do only what they're told to do. Both of these positions are obviously false, yet many innocent and unwary computerists have been lulled into believing them.

The real truth, which no computer journal has yet dared to reveal, is that computers are MAGICAL. Computer magic is both powerful and frightening, so the extent of the coverup is hard to determine.

Computer salesmen in their three-piece suits try to make computers look like just another tool in the hardware store. But listen to these instructions for a program: "Move, relative to the current position, to a new current position. This will change the current position by adding 5 to the current x position, and 9 to the current y position. There is no change to the screen." Does that sound to you like instructions on how to use a wrench or more like an incantation??

The deception is aided by the ordinary appearance of computer equipment. The screen looks like a TV. The keyboard is quite similar to a typewriter. The disks that are used for "programs" (or spells, to be more accurate) look like 45 rpm records.

Although appearances are deceiving, the minute you turn your computer on you should be able to tell it is NOT "just another tool." The first sound you hear is the beeping and bubbling of the cauldron starting to boil. Does your toaster make sounds like that when you turn it on?

Once the cauldron is simmering, you can put a disk into the computer drive, say a few magic words, and get an image to appear on the screen. Nothing too spooky here, until you take the disk out and the image won't disappear. No matter what you do to the disk the image will not get off the screen! Wouldn't you wonder what was up if the music kept on playing after you removed a record from your stereo turntable? Or if you took the slides out of a projector and the image stayed on the screen? Why hasn't anyone reported these computers to Ghostbusters?

Actually, almost all of the magic of computers rests in these disks. They are extremely sensitive critters, however, so you must be very, very careful with them if you want the magic to work. First and foremost, you should be aware that they don't like to be touched. This means that when you are not using them for a specific task, they should be kept in a paper cloak or sleeve. (The paper holds in the magic so it can't escape until needed.)

When you are ready to cast a spell, gently remove the disk from the sleeve. Hold the disk at the very edge and insert it into the "disk drive." The cauldron sits inside this so-called disk drive, and the disk must make contact with it before the magic can begin. To make good magic, the label on the disk must be facing the sky (in a horizontal disk drive), or facing left in a vertical disk drive.

What specific magic acts can these disks perform? They can draw, make music, compute income taxes (this one uses black magic), send letters to your friends, run a business or conjure up a recipe for fried bat wings.

So where does COMAL fit into this scheme of things? It is simply the most powerful magic around. ■

Learn Comal 2.0 with Rod The Roadman

by Borge Christensen

Ed. Note: Borge has created his own small universe with his ROD THE ROADMAN program. The program may be used by beginners quite easily. Since it uses a large grid and a visible robot (named ROD), it is ideal to use with a group of students. In addition to providing us with one of our best programs, Borge has thrown us a curve. His universe is created by a program that you can't list. But it allows you to merge your own additions to it! This is a fantastic feature. Now when you give the LIST command, you only will see the lines that YOU added. They are not mixed in with the lines that create the universe.

"How did Borge do that?", you may ask. Well, the answer is quite REVEALing. We leave you with that challenge. We'll even give you another hint: the procedure PROC.SHOWNAMES is very helpful. Next issue we will REVEAL the solution to you.

We think you can have hours of fun working with ROD THE ROADMAN. All you need is the COMAL 2.0 Cartridge and our TODAY DISK 9. Teachers and parents may find the system to be exactly what they were looking for. And why not? Borge is a teacher as well as the father of COMAL.

If you come up with problem screens you would like to share with our readers, send them in on a disk. We will collect them and make them available to others in the future. Watch future issues for an announcement.

Now, Borge tells us about ROD and his universe:

1. ROD AND HIS TOWN

Figure A shows a map of Rod's town. The horizontal lines are called roads, and the vertical ones are called streets. There are 10 roads and 15 streets. Rod is standing where 2nd street is crossing 2nd road. You can only see the peak of his cap. In Figure B, below, you can see Rod and the things in his world.

Figure B: Rod, Roadwall, Streetwall, Beeper

Then in Figure C you see what Rod's world may look like, when he has a house of his own built from streetwalls and roadwalls. Outside the house is a beeper. Rod is standing inside the house.

To get the computer to display Rod the Roadman and his world, insert the back side of TODAY DISK 9 into the drive and from COMAL 2.0 type:

```
RUN "ROD"
```

It takes a little while before Rod is ready. Then you get a picture like in Figure A. Now type these commands:

```
move
move
rt
move
lt
move
```

Here is a little explanation about the commands. As soon as Rod reads this one:

```
move
```

More ►

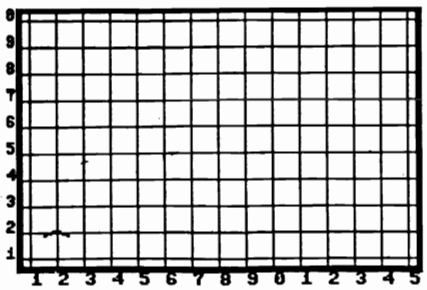


Figure A - Map of Rod's town.

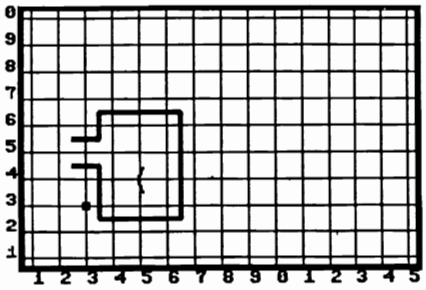


Figure C - Rod's house and a beeper.

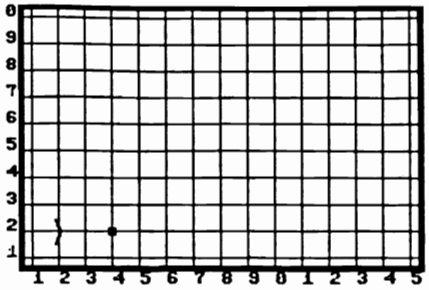


Figure 1 - Rod and beeper.

he moves forward to the next crossing. The command:

```
rt  (which can also be: right)
```

makes him turn to the right. The command:

```
lt  (which can also be: left)
```

makes him turn to the left.

Press <f7> to see the computer draw a fresh map of the town. Then enter this command line:

```
move;move;move;rt;move;move;move
```

Press <f7> again and enter the following command:

```
for i=1 to 10 do move;move;move;rt
```

Turn up the volume on the monitor and type:

```
putbeep
```

and watch Rod putting down a beeper. Then enter the commands:

```
move;rt;rt;move
```

to make him move away from the beeper and then back again. Note what happens as he comes back to the beeper. Now type the command:

```
pickbeep
```

and see what happens.

YOU NOW KNOW THAT:

move	moves Rod forward to next crossing
rt	turns Rod right 90 degrees
right	turns Rod right 90 degrees
lt	turns Rod left 90 degrees
left	turns Rod left 90 degrees
putbeep	makes Rod put a beeper down
pickpeep	makes Rod pick up a beeper

If Rod gets to a beeper, he can hear it. So he knows it is there.

EXERCISE 1.1

Shift to the textscreen by pressing <f8>. Then enter the command:

```
merge "problems"
```

to get some procedures loaded from the disk. Press <f7> to get a fresh map of the town. When it has been drawn type the command:

```
fig'1
```

You should now have a picture like Figure 1 on the screen. Now type a command line that makes Rod move the beeper to end up with a situation like the one shown in Figure 1B.

2. ROD THE ROADMAN MEETS BEN THE BRICKLAYER

In Rod's world there is a very clever fellow called Ben. He is a bricklayer. He decides where all the roadwalls and streetwalls are to be built. Sometimes he teases poor Rod by not telling him about the walls he sets up.

More ►

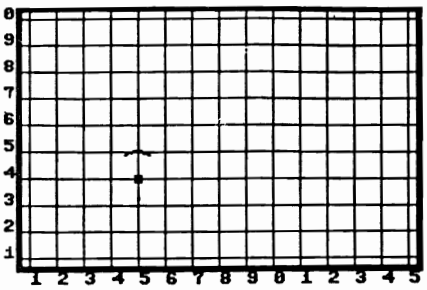


Figure 1B - Result for exercise 1.1

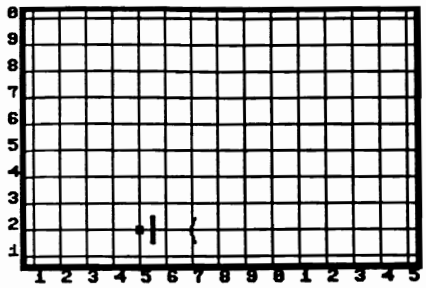


Figure 2 - Wall blocking Rod's path.

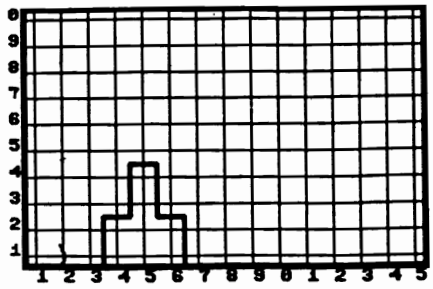


Figure 3 - Exercise 2.2

EXERCISE 2.1

Look at Figure 2. Press <f7> and enter the command:

```
fig'2
```

to get a similar picture on your screen. This is how the story behind it goes: Rod has put down a beeper where 5th street crosses 2nd road earlier that day. He is going back to pick it up again. But in the meantime Ben has built a roadwall across 2nd road between 5th street and 6th street. Now try to control Rod by using this command line:

```
move;move
```

Rod cannot pass the wall! The road is blocked! Help him to get around it by using the proper commands. Do not forget that in the end he has to pick up the beeper.

Rod cannot walk through or climb over the walls set up by Ben. He must make a detour. If he hits a wall, he laments in a most pitiful manner, because he has hurt himself.

It is not very often Ben teases Rod. Most of the time they help each other.

EXERCISE 2.2

Look at Figure 3. Ben has blocked an area of the town, because some dangerous remnants of gas have been found in the ground. He asks Rod to put a beeper where 5th street crosses 5th road, i.e. the crossing (5,5). The path Rod should follow is shown in Figure 3B.

Press <f7> and enter the command:

```
fig'3
```

Then type in some commands - preferably on one line - that make Rod do the job.

Sometimes it pays for Rod to memorize some commands, because he has to do them very often. This can be done by writing procedures for Rod to follow.

EXERCISE 2.3

Look at Figure 4. It shows a house that Ben has built for Rod. He is in his house now. Outside is a beeper with a lamp to light up the number of the house. Each morning this beeper is taken into the house, and each evening it has to be put out again.

Press <f7> and enter the command:

```
fig'4
```

Write some lines with commands to make Rod take the beeper into the house and put it in its upper right corner.

Press <f8> to get to the textscreen. Then press <f3> and watch COMAL writing a line number and the word:

```
proc
```

Add the name of the procedure to make the whole line look like this:

```
proc takein
```

Now press <RETURN> and enter these lines:

```
rt; move; move
lt; move; move
lt; move; move
lt; move
pickbeep
```

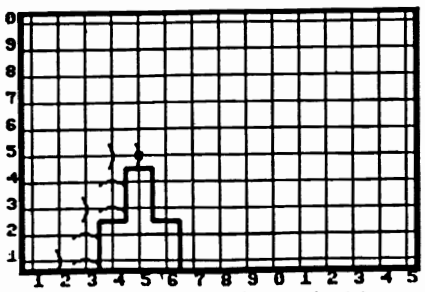


Figure 3B - Solution for exercise 2.2

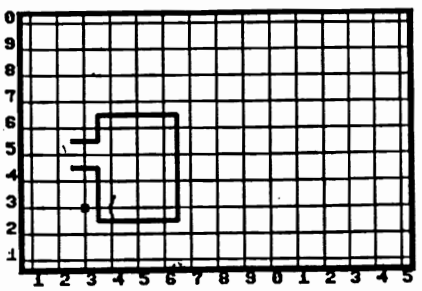


Figure 4 - Exercise 2.3

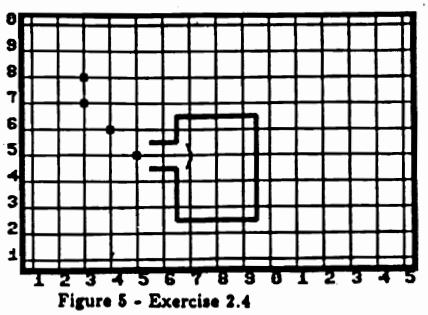


Figure 5 - Exercise 2.4

```
rt; rt; move
rt; move; move
rt; move; move
move; move; move
putbeep
```

Press <f4> to make COMAL finish the procedure with the statement:

```
endproc
```

NOTE: If two of you are working at the computer, it is best if one dictates while the other one does the typing. If you are working alone then use a ruler or the like to mark a line as you are typing it in.

Now press <f7> and type the commands

```
fig'4
takein
```

As you can see, the procedure is not in order. Rod was meant to put the beeper in the corner (6,6) and walk back to the place where he stood before. Look carefully at the picture (Figure 4) and the procedure in order to plan the necessary corrections and additions. Then press <f8> to get to the textscreen and correct the procedure. Clear the screen (press <SHIFT> with <CLR/HOME>). You can get a list of the present procedure by typing this command:

```
list takein
```

Having finished the corrections, press <f7> to get a fresh picture and enter the commands:

```
fig'4
takein
```

When the procedure is in order, write another one to make Rod take the beeper back to its place outside the house and then go back in again. Call the procedure takeout.

To test both procedures, press <f7> and use the commands:

```
fig'4
takein
takeout
```

EXERCISE 2.4

Look up Figure 5. Rod has got into his house but finds out that he has lost some of his precious little beepers on his way home. You should help him to pick them up and get back to his house again.

Shift to the textscreen by pressing <f8>. Press <f3> and start writing the procedure needed to make Rod pick up his lost beepers and then go back into his house again. Press <f4> to finish.

Press <f7> and enter the command

```
fig'5
```

Now test your procedure.

EXERCISE 2.5

Look up Figure 6. Rod only knows what right, left, and move mean. He knows nothing about up or down. Therefore he also do not know what stairs are all about. Ben has set up some walls to make him understand that. Rod is supposed to pick up the beepers on the stairs and end up on top of it all as shown in Figure 6B.

Write a procedure to help Rod do what he is supposed to. Use <f3> to start the procedure and <f4> to finish it. To test it, press <f7> and enter the command:

More ►

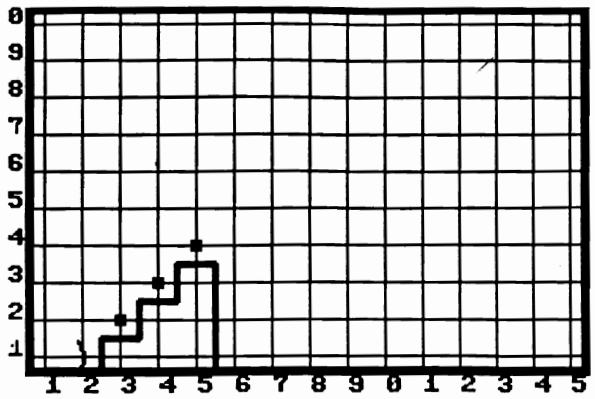


Figure 6 - Exercise 2.5

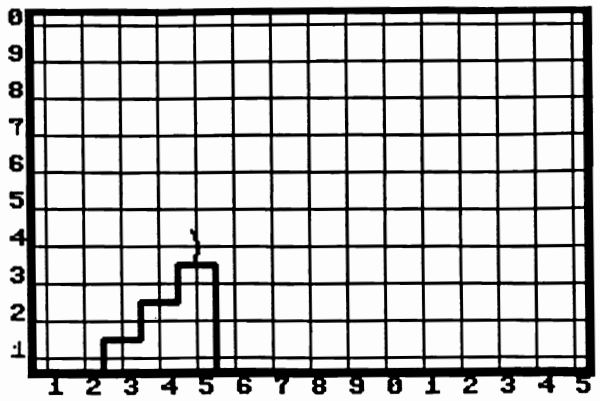


Figure 6B - Result for exercise 2.5

```
fig'6
```

to get the stairs displayed.

3. THERE IS A SYSTEM IN WHAT ROD DOES.

We have tried to make Rod follow a path that reminds you of some stairs. Let us take a closer look at what Rod does to pick up the beepers on each of the stairs.

EXERCISE 3.1

Press <f7>. When the town has been drawn type:

```
fig'6
```

and check that a picture like Figure 6 is coming up on the screen. We shall try to make Rod do the same as in exercise 2.5.

Press <f8> to get the textscreen. Then press <f3> to start a procedure and type:

```
proc one'step
  lt; move
  rt; move
  pickbeep
```

Press <f4> to finish the procedure.

Press <f7> to get a map of the town and enter the command

```
fig'6
```

Now type the command:

```
one'step
```

and watch what happens. Go ahead using the same command until a picture like Figure 6B is on the screen.

Press <f8> to get the textscreen and enter a new procedure stairs that will make Rod pick up all the beepers as he did before. Test it by pressing <f7> and typing the commands:

```
fig'6
stairs
```

Very often we can help Rod best by giving the problems a second thought, before trying to have them solved systematically. As a rule it is not a good idea to start chasing Rod around without pondering first to find out what we really want him to do.

Ed. Note: Ah, yes. While Borge was in Madison, he left us a challenge. Can you write a procedure the will get ROD out of any house no matter where he is? Those who heard his talk at the MARCA computer show in July already saw the solution - but can you do it on your own now? Send your solution to us right away! We will print the names of all who can solve Borge's challenge. To do this you should use the scanners that ROD always carries with him. CLEAR'AHEAD will return TRUE if there is no obstacle ahead of ROD. Likewise, CLEAR'LEFT and CLEAR'RIGHT will check on either side. ■

Comparing Disk Files

by Len Lindsay

Did you ever make a copy of a file, and then wish you could verify that the copy was really identical? Did you ever have two files on two different disks, and wonder if they were the same? We have those same kind of problems here. So I wrote a short COMAL 2.0 program that will copy a file, and then verify it. It works with single drives as well as dual drives, but the files must be less than 28K long (all the memory left free).

This program not only compares two files, but prints the characters (and their ASCII values) in both files side by side - pointing out any mismatches! Non printing characters are printed as a period.

You can use this program to make a duplicate copy of a file on the same disk as the original file. Just use a different name for the target file.

If you use two disks with a single drive, you will have to swap disks (the program tells you when to swap the disks). If both files are on the same disk, just keep that disk in the drive the whole time - hit RETURN at each prompt to insert a disk.

IMPORTANT NOTE: To copy a PRG type file, you must include a ",PRG" as the last part of the target filename: **MENU.BACKUP,PRG**

POSSIBLE VARIATIONS

Printing the contents of the files on the screen takes time. If you don't wish to see the contents displayed (and thus speed up the process), just add the following line as the first line in the procedure named PRINTOUT:

```
RETURN
```

If you want to copy files but not compare them, just delete the line that calls the COMPARE procedure as well as deleting the whole COMPARE procedure itself:

```
DEL COMPARE
DEL 50 //as in the listing numbered by 10
```

If you want to just compare files (no copying), delete the COPY'FILE procedure and the line that calls it:

```
DEL COPY'FILE
DEL 40 //as in the listing numbered by 10
```

The program does not check if the files exist before comparing them. You may wish to add the **FILE'EXISTS** function (listed in COMAL TODAY #6 page 42) or use the **TRAP** structure. But remember, the more you add to the program, the less memory will be free for copy and compare storage.

```
//delete"0:copy/compare" // comal 2.0 only
//save"0:copy/compare"
init
copy'file(source$,target$,max'size)
compare(source$,target$,max'size)
//
PROC init
  PAGE
  max'size:=28*1024
  DIM source$ OF 23, target$ OF 23
  INPUT "Source Name: ": source$
  INPUT "Target Name: ": target$
ENDPROC init
//
PROC copy'file(source$,target$,max) CLOSED
  DIM image$ OF max, dummy$ OF 0
  read'it'in
  IF LEN(image$)>=max THEN END "Too long"
  INPUT "Insert target disk:": dummy$
  write'it'back
  //
```

- More ►

```
  PROC read'it'in
    OPEN FILE 5,source$,READ
    WHILE NOT EOF(5) DO
      image$:+GET$(5,254)
    ENDWHILE
    CLOSE
  ENDPROC read'it'in
  //
  PROC write'it'back
    MOUNT
    length:=LEN(image$)
    OPEN FILE 5,target$,WRITE
    count:=1
    WHILE count+253<length DO
     PRINT FILE 5:image$(count:count+253),
     count:+254
    ENDWHILE
    PRINT FILE 5: image$(count:length),
    CLOSE
  ENDPROC write'it'back
ENDPROC copy'file
//
PROC compare(name1$,name2$,max) CLOSED
  DIM image$ OF max,text$ OF 1,dummy$ OF 0
  OPEN FILE 2,name2$,READ
  WHILE NOT EOF(2) DO
    image$:+GET$(2,254)
  ENDWHILE
  CLOSE FILE 2
  INPUT "insert source disk:": dummy$
  OPEN FILE 3,name1$,READ
  errors:=FALSE; byte:=0
  WHILE NOT (EOF(3) OR byte=LEN(image$)) DO
    byte:+1
    text$:=GET$(3,1)
    IF text$<>image$(byte) THEN errors:+1
    printout
  ENDWHILE
  PRINT "Errors found:";errors
  IF NOT EOF(3) THEN
    PRINT "File";name1$;"is longer."
  ELIF byte<LEN(image$) THEN
    PRINT "File";name2$;"is longer."
  ELIF NOT errors THEN
    PRINT "The files matched."
  ENDIF
  CLOSE
  //
  PROC printout
    //RETURN//add this line to skip print
    PRINT USING "#####>": byte;
    print'char(ORD(image$(byte)))
    print'char(ORD(text$))
    IF errors THEN PRINT "Mismatch";
    PRINT // carriage return
  ENDPROC printout
  //
  PROC print'char(n)
    IF (n>=32 AND n<128) OR n>=160 THEN
      PRINT CHR$(n);
      PRINT USING "###": n;
    ELSE
      PRINT USING ". ###": n;
    ENDIF
  ENDPROC print'char
ENDPROC compare
```

Unit to Unit File Copier

This small program can copy programs using two 1541 drives. One of the drives must be set to device 9. It displays the directory of the disk in the unit 8 drive. Then it asks for the filename. To copy a PRG type file include ",PRG" after the file name. The file is then written to drive 2 (unit 9 drive 0 is referred to as drive "2:"). Unit 9's disk directory is then displayed. Easy! This is the whole program:

```
// copy files unit to unit - max 14K
DIR "0:*"
INPUT "File name: ": name$
OPEN FILE 2,"0:"+name$,READ
OPEN FILE 3,"2:"+name$,WRITE
PRINT FILE 3: GET$(2,14000),
CLOSE
DIR "2:*"
```

Also see a related program on page 42.

Text Package

by Dick Klingens

When we created a large monitor program with a lot strings in which we stored a help menu, we were not able to extend that program with more disk operations, because all memory was occupied.

Two possibilities were left; leaving out the help menu or storing the help strings in another part of the memory.

We did the latter. We created a RAM disk (a text buffer) as a package and we called that package TEXT.

The new package has 4 procedures and one function. You will notice that the commands are similar to the file commands of Pascal and work in much the same way.

PROC readln(REF x$)
PROC writeln(REF x$)
PROC reset
PROC rewrite
FUNC eot

READLN fetches a string from the buffer. During this fetch there is a test on reading the end of the buffer. If so, an error message is printed and the program is stopped.

WRITELN does the reverse. It writes a string into the buffer. If the 16K buffer is full, the message 'out of memory' is printed.

RESET directs the reading pointer to the first position in the buffer. This statement can be used to read again from the beginning.

REWRITE directs the reading and writing pointer to the first buffer position. It empties the buffer!

EOT (End Of Text) is a function that returns TRUE when the reading pointer is in the same position as the writing pointer. If EOT=TRUE, then there is no more text in the buffer. This function is similar to the EOD function built into COMAL.

The following example shows how to use this package.

```
USE text
rewrite // empties buffer
DIM x$ OF 40
PRINT "Enter any text. Press RETURN on"
PRINT "a blank line to end."
REPEAT
  INPUT x$
  IF x$<>"" THEN writeln(x$)
UNTIL x$=""
reset // read pointer to first position
WHILE NOT eot DO
  readln(x$)
  PRINT x$
ENDWHILE
END "All done."
```

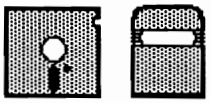
This package is valuable to programmers who need access to lots of text without using the disk drive. One use might be in a bulletin board program to speed up menu printing.

The source code for this package is too long to list, so it and the assembled package are on *Today Disk #11*. **Special note:** The *DEMO/TEXT2* program also on the disk shows that any text in the RAM disk buffer is also saved with the program.

More ►

```
; src.text (comal module)
; by m.bokhorst, nov85
; revised by d.klingens
; dutch comal users group
;
;- variables & constants -
defpag = %01000110
dummy = $ca2f
proc = 112
endprc = 126
func = 227
endfnc = 126
pshint = $c9ce
str = 2
ref = 117
point = $fb
fndpar = $c896
copy1 = $45
copy2 = $47
copy3 = $49
copydn = $c8a2
runerr = $c9fb
;
;-- module --
;
* = $8009
 .byte defpag
einde .word end
 .word dummy
 .byte 4,'text'
 .word procs
 .word reset
 .byte 0
;
;-- procedures
;& functions --
procs .byte 7,'rewrite'
 .word hempty
 .byte 7,'writeln'
 .word hput
 .byte 6,'readln'
 .word hget
 .byte 5,'reset'
 .word hres
 .byte 3,'eot'
 .word heot
 .byte 0
;
;-- headers --
;
hempty .byte proc
 .word empty
 .byte 0
 .byte endprc
;
hput .byte proc
 .word put
 .byte 1
 .byte str+ref
 .byte endprc
;
hget .byte proc
 .word get
 .byte 1
 .byte str+ref
 .byte endprc
;
hres .byte proc
 .word reset
 .byte 0
 .byte endprc
;
heot .byte func
 .word eot
 .byte 0
 .byte endfnc
;
;-- code --
;
empty lda #<end
 ldy #>end
 sta einde
 sty einde+1
;
reset lda #<end
 ldy #>end
 sta point
 sty point+1
 rts
;
eot jsr teof
 lda #0
 rol a
 tax
 lda #0
 jmp pshint
;
put lda #1
 jsr fndpar
 lda copy1
 clc
 adc #<2
 sta copy1
 lda copy1+1
 adc #>2
 sta copy1+1
 lda einde
 ldy einde+1
 sta copy2
 sty copy2+1
 ldy #1
setup lda (copy1),y
 sta copy3,y
 dey
 bpl setup
 jsr len
 lda copy3+1
 clc
 adc einde
 tax
 lda copy3
 adc einde+1
 cmp #$c0
 bcs out
 stx einde
 sta einde+1
 jmp copydn
;
eof ldx #201
 .byte $2c ;skip 2
out ldx #52
 jmp runerr
;
teof lda point
 sec
 sbc einde
 lda point+1
 sbc einde+1
 rts
;
get jsr teof
 bcs eof
 lda #1
 jsr fndpar
 lda copy1
 clc
 adc #<2
 sta copy2
 lda copy1+1
 adc #>2
 sta copy2+1
 ldy #1
 lda (copy1),y
 sec
 sbc (point),y
 dey
 lda (copy1),y
 sbc (point),y
 bcc noroom
 lda point
 ldy point+1
 sta copy1
 sty copy1+1
 ldy #1
setup1 lda (point),y
 sta copy3,y
 dey
 bpl setup1
 jsr len
 lda point
 clc
 adc copy3+1
 sta point
 lda point+1
 adc copy3
 sta point+1
 jmp copy
noroom lda (copy1),y
 sta copy3
 pha
 iny
 lda (copy1),y
 sta copy3+1
 pha
 lda point
 ldy point+1
 sta copy1
 sty copy1+1
 jsr len
 ldy #1
 lda (point),y
 clc
 adc point
 tax
 dey
 lda (point),y
 adc point+1
 tay
 txa
 clc
 adc #<2
 sta point
 tya
 adc #>2
 sta point+1
 ldy #1
 pla
 sta (copy1),y
 dey
 pla
 sta (copy1),y
copy ldx copy3
 lda copy3+1
 tay
 beq l001
 eor #255
 tay
 iny
 clc
 lda copy1
 adc copy3+1
 sta copy1
 bcs l002
 dec copy1+1
l002 clc
 lda copy2
 adc copy3+1
 sta copy2
 bcs l003
 dec copy2+1
l003 lda (copy1),y
 sta (copy2),y
 iny
 bne l003
 inc copy1+1
 inc copy2+1
l001 dex
 bpl l003
 rts
;
len lda copy3+1
 clc
 adc #<2
 sta copy3+1
 lda copy3
 adc #>2
 sta copy3
 rts
end .end
```

Instructional Videos

by Garrett A. Hughes 

As a teacher, I have found that creating lessons and recording them on video tape is one of the most exciting ways to utilize C64 COMAL. The *"instructional videos"* that I produce are intended for viewing by my student audience.

To make a lesson, I use a standard video cassette recorder, a microphone, and a light pen in conjunction with C64 COMAL 2.0. While making a lesson, I employ the monitor screen as a *"blackboard"*. Instead of writing on the blackboard with chalk, I type to the screen from the keyboard, and draw on the screen with the lightpen. As I draw or type, I describe what I'm doing as if I were teaching a lesson to one of my classes.

Recording the lesson on video tape is easy. You need only connect the RF output on the back of the C-64 to the RF input on your video recorder. Connect your microphone to the audio input of the same VCR (use an audio amplifier if necessary). Start the recorder when you begin the lesson and presto, you have a permanent record of whatever you desire to teach.

The video tapes you make can be duplicated and kept in the school library. They can be checked out overnight like books on reserve or can be used in your school's audio-visual center. Students can use them to catch up on work they have missed, review material they didn't quite understand in class, or take a peek at more advanced material. Substitute teachers can use the tapes in your classes while you're away.

And that's just the beginning. Suppose a video lesson deals with the use of a piece of software that runs on the C64. First, load that software into the C64, then have the student play your video tape with the tape machine's outputs leading to the C64's monitor. Hitch the output of the C64 to the input of the VCR as described earlier. When the student wants to hear and see the lesson, he/she presses the PLAY button on the VCR. When the student wants to try what they are observing they press the STOP button on the VCR. The same software that you are describing will now be lighting up the monitor screen. Yes, really! **Just pressing the PLAY or STOP button on the VCR shifts the student from a passive observer to an active participant or vice versa.**

When making a video, I use text and graphics screens that I have prepared ahead of time and stored on disk. The COMAL PACKAGES make it a cinch to prepare just the screen I want, and to retrieve it at the appropriate time.

With COMAL's lightpen and sprite packages, I can create images that move to any location on the screen to which I point. The effect is striking and a minimum of programming effort is involved. For example, suppose you have a graphics picture you want to talk about. You can create a sprite *"pointer"* and move the pointer about the screen under lightpen control. You can put the pointer wherever you want on the graphic image. The video viewer is directed by the pointer to the exact location on the screen that you are talking about.

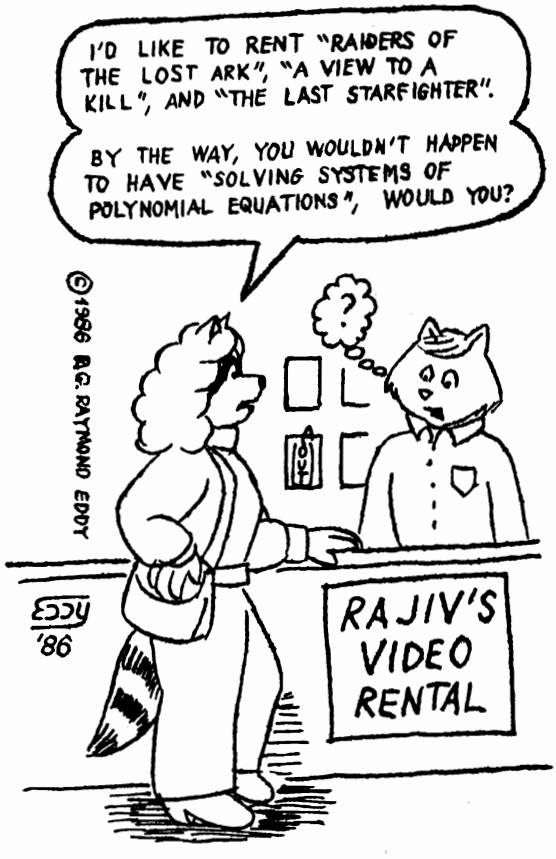
The best feature of instructional videos, which you prepare yourself, is that they allow you to take advantage of your own years of teaching experience.

More ►

That's something that a lesson under program control, or a commercially prepared videodisk can never duplicate. Another advantage of this system is that the price of all the components is affordable, and many of your students will already have access to a VCR.

Give it a try. I can almost guarantee that the response to your efforts by your students and school officials will be overwhelmingly positive.

[Ed. Note: This is a fabulous method for preparing lessons! The same techniques could be used for COMAL 0.14 programs, or any other language. Garrett teaches at Burlington High School in Burlington, Vermont 05401]. ■



VAL and STR$ 0.14

by Dick Klingens

Many different procedures and functions have been written to emulate the VAL and STR$ commands left out of COMAL 0.14. Each of these have either been simple and short (not doing fractional numbers or handling negative numbers) or complex and long (taking up more memory).

The following two procedures work using any Commodore compatible disk drive (1541, MSD, etc.), provide full conversion of numbers and strings, and take up little space.

Each routine uses a *"buffer"* in the disk drive's memory. First the buffer is opened and the number or string is printed to it. Then the buffer pointer is reset to the beginning and the number or string is read back in (in the desired format). The final step is to close the file and return the result. There doesn't even have to be a disk in the drive (although the drive does have to be turned on).

```
func val(x$) closed
  open file 100,"#",unit 8,2,read
  print file 100: x$ // print to file
  pass "b-p:2,1" // reset to beginning
  input file 100: y // bring back in
  close file 100 // new form and close
  return y
endfunc val
//
proc str(x,ref y$) closed
  open file 100,"#",unit 8,2,read
  print file 100: x // print to file
  pass "b-p:2,1" // reset to beginning
  input file 100: y$ // bring back in
  close file 100 // new form and close
endproc str
```
■

Fractals

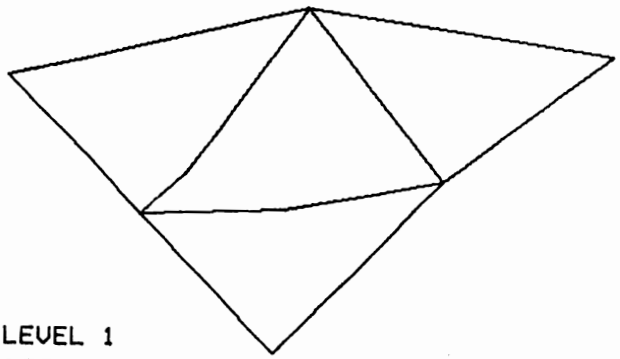
by Kevin Quiggle 

This program uses the 1520 plotter or the graphics screen to display fractal images. The plotter display is much more impressive.

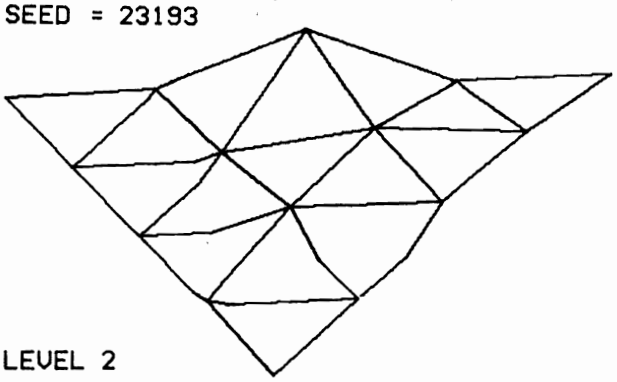
Three dimensional fractal diagrams are drawn at up to seven levels of complexity. You get to choose the complexity level and the seed (for the random number generator). If you use the same seed twice, you should get the same results each time. A different seed should give a different fractal diagram. If you don't enter a seed, the program will select one for you.

When the display if finished, the plotting time is also displayed. It takes nearly 4 1/2 hours to plot a level 7 fractal on the 1520 plotter.

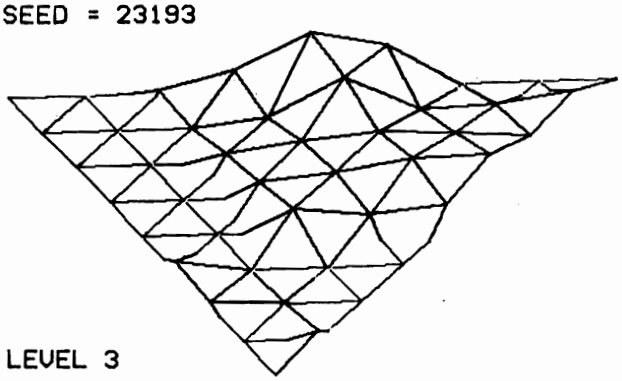
Fractals are a very interesting subject. They are a sophisticated way to generate 3D graphics. It has been reported that Lucasfilm has used fractals in every computer game they've produced (the Epyx game *Koronis Rift* for example). Fractals were also used in the second *Star Trek* movie. Ask at your local library for some reference articles about fractals. Scientific American had an in depth article a few years ago.



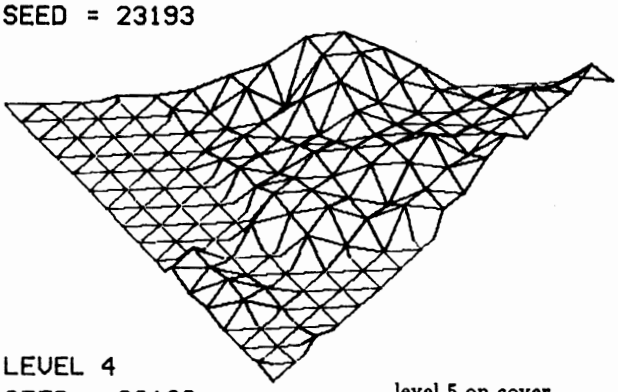
LEVEL 1
SEED = 23193



LEVEL 2
SEED = 23193

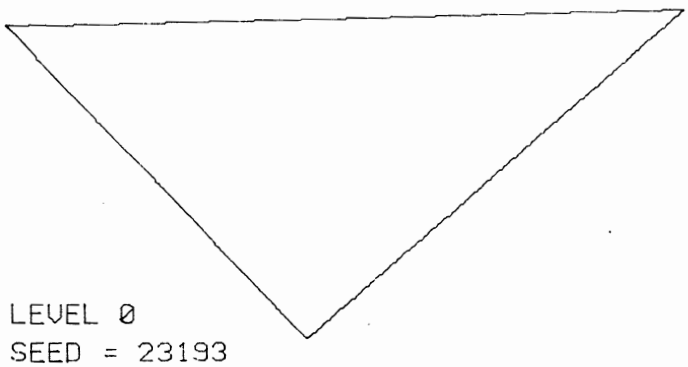


LEVEL 3
SEED = 23193



LEVEL 4
SEED = 23193

level 5 on cover



LEVEL 0
SEED = 23193

More ►

Integrated Software

Several COMAL 2.0 Cartridge users have discovered that the cartridge includes a small integrated system of built-in programs, including Modem, Word Processor, Data Base and Picture Printer.

MODEM TERMINAL

Issue the command:

select output "sp:"

Now all your normal output is directed through a normal Commodore modem. You can even have automatic true ASCII conversion by adding the attribute "/a+":

select output "sp:/a+"

You also can switch from the keyboard as the input location, to the modem using this command:

select input "sp:"

WORD PROCESSOR

We now regularly get letters using the cartridge's built in word processor. This is how it works:

Type anything you want on the screen. Use the *STOP* key to get to the next line. Other commands include:

CONTROL K - Erase to end of line
CONTROL L - Go to the end of the line
CONTROL B - Move back one word
CONTROL F - Move forward one word

Plus, the cursor keys work, as do the *INSERT* and *DELETE* keys.

When the screen looks right, just press *CONTROL P* and the textscreen is printed on your printer.

DATA BASE

James White sent us the following note (using the built in Word Processor mentioned above):

COMAL is great. It even has a built in Data Base. The way it works is with the command **FIND**. If you set up a program file for your database and enter the data as a comment line (//), then **FIND** will seek the string you search for. You can use **DEFKEY** to set the function keys to expedite the search. If you set up portions of the data as a **PROC** you can limit the search to only that portion. Here is an example of how you might set it up. The great thing is that the program doesn't take up much memory, leaving the rest for data.

```
0010 USE system
0020 defkey(1,"find"+CHR$(34))
0030 defkey(3,"find books "+CHR$(34))
0040 //James White 505-982-0567
0050 //Captain COMAL 608-222-4432
0060 PROC books
0070   //Moby Dick  Herman Melville
0080   //Hamlet     William Shakespeare
0090 ENDPROC books
```

PICTURE PRINTER

If you have a Commodore 1525, MPS801, MPS803, or compatible, all you need to do is press *CONTROL D* and the full graphic screen (multi-color or hi-res) is printed on your printer.

Further Reference:

Modem Fun With COMAL 2.0, COMAL Today #9, page 10
Amazing Delete Key, COMAL Today #7, page 19
COMAL 2.0 Auto ASCII Conversion, COMAL Today #6, page 40 ■

COMAL Structures While Loops

by Richard Bain

Programmers who are new to COMAL will find many new programming structures to make programs easier to write and understand. The **WHILE** loop is one of these. Here are some hints on when to use the **WHILE** loop, how to use it, and what similar structures may be used instead.

The **WHILE** loop should be used in cases when the programmer or user would think in English, *"While this is happening, I should be doing something."* Let's see how this might look in a few real programming situations:

```
while not eod do
  read number
  print number
endwhile
```

This program fragment will read numbers from data statements and print them out. It automatically exits the loop when there is no more data to read. It could easily be changed to do almost anything else with the numbers, but the important point to remember is that the amount of numbers does not matter. Data statements can be added or deleted, but the **WHILE** loop doesn't need to be changed.

```
while raining do play'inside
```

This is an example of a one line **WHILE** loop. The differences between this loop and the loop above are that you can only use one statement after the **DO**, and you don't type in **ENDWHILE**. In this example, raining can either be a simple variable or a function call which **returns** a 1 for **TRUE** or a 0 for **FALSE**. Play'inside is a **procedure** call. This may be a menu to let you choose one of many computer games, or it could just be the name of your favorite game. That's up to you. Even though play'inside is only one statement, being a **procedure** call allows it to do many things. When it stops raining, you will no longer be nagged to stay inside. See, it's almost like English.

```
while key$ = chr$(0) do null
```

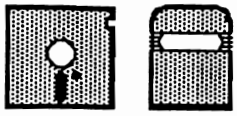
This is a useful line which you may already have seen in several programs. It is simply a delay loop. Usually, a message will first be printed on the screen to tell the user to hit a key to continue. Then this loop makes the program pause until a key is pressed. This is useful in many cases such as a text reading program in which the user will want to pause before unread text scrolls off the screen. Did you know that doing nothing (**NULL**) could be so useful?

Now, let's compare the above examples to similar examples with other structures. The first one could be replaced with a **FOR** loop in many cases. You would want to do this if you knew exactly how many numbers to read in, or if you wanted to store the data in an *array* and needed a counter for the *array* index.

In the second example, many people might be tempted to use a **REPEAT** loop. This is not a good idea. What if it isn't raining at all. The **REPEAT** loop will always make you play inside for a little since a **REPEAT** loop *always* executes at least once. The **WHILE** loop will be skipped over completely if it isn't raining.

BASIC only has a **FOR** loop. Some experienced BASIC programmers will simulate **WHILE** and **REPEAT** loops with those awful **GOTO** statements, but in COMAL you don't need to. Now, aren't you glad you switched to COMAL. ■

Program Outliner



by Len Lindsay

"Aldebaran's $97 utility, Source Print, offers a structured outline format feature that automatically draws connecting lines between program block elements."
-- PC Magazine, Sept 16, 1986, page 63

While trying to keep up to date, I manage to read literally dozens of computer magazines and newsletters each week. The notice reprinted above caught my eye. Program outlining is a very nice idea. Why not give *COMAL Today* readers a program outliner?

I sat down, and in 1 hour had a program that would outline both COMAL 0.14 and COMAL 2.0 programs that were LISTed to disk. The COMAL 0.14 programs must be listed with indentations (see page 3 for the modification to COMAL). The program is listed below - output from itself! It doesn't have the fancy features of the IBM PC program, but it is short and readable.

```
    // delete "program'outliner"
    // save   "program'outliner"
    //  by Len Lindsay
    DIM space$ OF 3, line$ OF 3
    DIM top$ OF 3, bottom$ OF 3
    DIM middle$ OF 3, filename$ OF 20
    DIM out$ OF 20, this'line$ OF 120
    DIM next'line$ OF 120
    init
    printout
    //
==> PROC init
!       disk'indent:=0
!       // set at first structure top
!       space$:="   " // 3 spaces
!       line$:="!  " // ! and 2 spaces
!       top$:="==>"
!       middle$:="+->"
!       bottom$:="-->"
!       PRINT CHR$(147),CHR$(14)
!       PRINT TAB(8),"Program OUTLINE Processor"
!       PRINT
!       PRINT "This program takes a listed program on"
!       PRINT "disk and prints it to the screen (ds:),"
!       PRINT "printer (lp:), or disk file (0:filename)",
!       PRINT "with the different structures outlined."
!       PRINT
!       INPUT "listed program filename? ": filename$
!       INPUT "output location? ": out$
--> ENDPROC init
==> PROC printout
!       OPEN FILE 2,filename$,READ
!       SELECT OUTPUT out$
!       INPUT FILE 2: this'line$
!       fix'line(this'line$)
!       this'indent:=indent(this'line$)
!       last'indent:=this'indent
!   ==> WHILE NOT EOF(2) DO
!   !       INPUT FILE 2: next'line$
!   !       fix'line(next'line$)
!   !       next'indent:=indent(next'line$)
!   !       preline
!   !       PRINT this'line$
!   !       this'line$:=next'line$
!   !       last'indent:=this'indent
!   !       this'indent:=next'indent
!   --> ENDWHILE
!       next'indent:=indent(this'line$)
!       preline
!       PRINT this'line$ // last line in file
!       CLOSE FILE 2
!       SELECT OUTPUT "ds:"
--> ENDPROC printout
    //
==> FUNC indent(REF text$) // first space is not indent
!       level:=0
!   ==> WHILE LEN(text$)>1 DO
!   !   ==> IF text$(2:2)=" " THEN
!   !   !       level:+1
!   !   !       text$:=text$(2:LEN(text$))
!   !   +-> ELSE
!   !   !       RETURN level
!   !   --> ENDIF
!   --> ENDWHILE
!       // text$:=text$+"//" // optional tack on blank
!       RETURN this'indent // blank line at same level
--> ENDFUNC indent
    //
==> PROC fix'line(REF text$) // remove line number
!       text$:=text$(5:LEN(text$))
--> ENDPROC fix'line
    //
==> PROC preline
!       prefix
!   ==> IF this'indent<last'indent AND this'indent<
        next'indent THEN // wrap line
!   !       PRINT middle$,
!   +-> ELIF this'indent<next'indent THEN
!   !       PRINT top$,
!   +-> ELIF this'indent<last'indent THEN
!   !       PRINT bottom$,
!   +-> ELSE
!   !       PRINT space$,
!   --> ENDIF
--> ENDPROC preline
    //
==> PROC prefix
!   ==> IF disk'indent THEN
!   !   ==> FOR temp=1 TO this'indent STEP disk'indent DO
!   !   !       PRINT line$;
!   !   --> ENDFOR temp
!   +-> ELSE
!   !       disk'indent:=next'indent-this'indent
!   --> ENDIF
--> ENDPROC prefix ■
```

Introduction to Procedures

by David Stidolph 

The process of writing a program can be described as defining the problem and breaking it into small, manageable, separate sections. With this method, a complex program can be designed as a series of small sections, called procedures and functions.

A procedure is a block of program statements, called by name, which perform a useful task. The procedure can be called from anywhere in the program. This saves memory because it is only defined once, not every time it is needed. Another advantage of procedures is that once the program has been **RUN**, the procedure may be called from direct mode. This powerful feature lets you add command words to COMAL (while the procedure is in memory). The following example shows a very simple procedure definition and how it is called.

```
0010 print "I am a regular statement."
0020 print'procedure
0030 print "This is after the call."
0040 end
0050 //
0060 proc print'procedure
0070   print "Inside the procedure."
0080 endproc print'procedure
```

When **RUN**, the program prints:

I am a regular statement.
Inside the procedure.
This is after the call.

To see how **print'procedure** has become a command, call it from the immediate mode. Just type:

```
print'procedure
```

The response is:

Inside the procedure.

Though this procedure is simple, it shows how to define and call a procedure. Line 60 is the procedure header. It defines the name of the procedure, and what information, if any, needs to be passed to it (more on this later). Line 80 marks the end of the procedure. When this statement is reached, the procedure is finished and control returns to the line following the procedure call. Line 20 shows how to call a procedure - by using its name. In this case the procedure is called **print'procedure**. Notice that the sample output shows how control passes to the procedure and back to the statement following the call.

But why go through all the trouble and typing just to print three lines? Procedures are most useful when called from several points in a program. The following procedure will clear the keyboard buffer. This may not seem very important, but many programs spend time drawing graphics, doing file access, or computing some formula. During this time the user may press a key on the keyboard by mistake. It is best to clear the keyboard buffer before you do any **INPUT**.

```
9000 proc clearkeys
9010   while key$>chr$(0) do null
9020 endproc clearkeys
```

As important as procedures are, typing each procedure I needed into every program would require more time than I spend designing the entire program. COMAL gives me the ability to maintain a procedure library. I **LIST** my procedures to disk and **ENTER** them into programs as I need them - without retyping the procedure.

More ►

Notice that I numbered the lines of this procedure starting at 9000. There is a reason for this. In COMAL 0.14, lines merged from disk overwrite identically numbered lines in memory. If I numbered **clearkeys** to start at line 10 as in the previous example, it would replace lines 10-30 of the program it was entered into. As a rule, number the procedure to start at line 9000 before listing it to disk for your procedure library.

To list **clearkeys** to disk, type it in and use the following command:

```
list "clearkeys.proc"
```

COMAL will now list the program lines in memory into a file called *clearkeys.proc*. I put the *.proc* after the name *clearkeys* so that I will remember this is a listed procedure. When I want to merge this procedure with another program I have in memory, I will use the commands:

```
enter "clearkeys.proc"
renum
```

The **RENUM** command forces line numbers to start at 10 and then increment in units of 10. After the **RENUM** you may merge in a second procedure that also is numbered from 9000 without any conflicts. A variation of the **RENUM** command can make line numbers start at 9000 to let you list a procedure to disk:

```
renum 9000
```

I strongly advise you to always break your program into small procedures. It makes it easier for you to debug your program, and makes your program more readable. Make sure you use meaningful names for your procedures. **Convert'dollars'to'yen** is easier to understand than **cvtdy**. Since COMAL puts all variable names into a table and tokenizes them within the program, virtually no extra space is wasted by long names. I also suggest you keep your procedures short enough to see the entire procedure on the screen at once.

FUNCTIONS

Functions are like procedures except they return values. A function call looks like a variable, but acts like a constant. For example, it can appear on the right side of an assignment statement (:=), but not on the left. Function values may be printed or used in tests for the **IF**, **CASE**, **WHILE**, and **UNTIL** statements. A function call may not stand alone on a line like procedure calls can, because the value of the function would have nowhere to go.

You have been using functions in your programs already if you have used **ABS**, **COS**, **RND**, **PEEK**, or any other built in function. COMAL 0.14 lacks the built in function of **PI**, so we will define it here:

```
func pi
  return 3.14159266
endfunc pi
```

Notice that I did not print the line numbers. From here on I will just print the statements - you can provide line numbers with the **AUTO** command. When called, **pi** returns the value 3.14159266. The following program uses **pi** to calculate the area of a circle. (If you are typing this in from COMAL 2.0 where **PI** is a built in function, you do not need to include the last four lines.)

```
input "Enter radius of circle: ": radius
print "Area of the circle is";
print pi*radius^2
//
```

More ►

```
func pi
  return 3.14159266
endfunc pi
```

PARAMETERS

Parameters are values or variables you pass to the procedure or functions. For example, **PRINT RND(1,10)** will print a random number between 1 and 10. In this case the numbers 1 and 10 are the parameters for the **RND** function.

The program above figures the area of a circle. The function below will return the area of a circle from a given radius:

```
func circle'area(radius)
  return pi*radius^2
endfunc circle'area
```

Radius is the parameter of the function circle'area. When you call circle'area you must put a number, or numeric variable, inside of parentheses right after the name. The following is the find area program rewritten to use the circle'area function.

```
input "Enter radius of circle: ": value
print "Area of the circle is";
print circle'area(value)
//
func pi // don't type this function in
  return 3.14159266 // for COMAL 2.0
endfunc pi
//
func circle'area(radius)
  return pi*radius^2
endfunc circle'area
```

I changed the name of radius in the program to value to show you that the variables you pass to procedures or functions don't need to have the same name. The parameter name in the procedure or function header is just an easy way to refer to the value being passed. If you do happen to use the same name in the procedure header, there is no problem. COMAL will automatically create a new variable using the name in the header and use that definition only until the procedure or function ends.

Parameters can be of any standard variable type (real, integer, string, or array). The main purpose of parameters is to make procedures and functions more versatile. For example, a poor way to write the program above could be:

```
input "Enter radius of circle: ": value
print "Area of the circle is";
print circle'area
//
func pi // don't type this function in
  return 3.14159266 // for COMAL 2.0
endfunc pi
//
func circle'area
  return pi*value^2
endfunc circle'area
```

In this case, there is no parameter; this function is specific to this program. You always have to set the variable value to the radius of the circle before calling function circle'area. This is far less transportable than the original.

REFERENCE PARAMETERS

Suppose we make a general purpose procedure to swap the values of two variables.

```
proc swap(first,second) closed
  temp:=first
  first:=second
  second:=temp
endproc swap
```

More ►

Here is some code to test the procedure:

```
a:=1; b:=2
swap(a,b)
print "a=",a
print "b=",b
```

When **RUN**, you find that after **swap**, **a** is still **1** and **b** is still **2**. The problem is that when **swap** was called, it created the variables **first** and **second**. When the procedure ended, their values disappeared. Variables **a** and **b** were never changed. Reference parameters must be used in the **swap** procedure. If changes are made to a reference parameter, the changes are also made to the variable that it represents (even if they have different names). Here is the correct procedure:

```
proc swap(ref first,ref second) closed
  temp:=first
  first:=second
  second:=temp
endproc swap
```

Now if you use the same calling code, variable **a** will equal **2** and variable **b** will equal **1** after calling **swap**. You should make parameters referenced if you want to change their values within the procedure. One word of caution: only use the reference notation when you need it. If you make a parameter referenced, you will find you can't pass it a constant (or function value). For example, the following command would give you an error with the swap procedure above.

```
swap(5,9)
```

BUILDING BLOCKS

Procedures and functions are the building blocks of a program. They allow you to *borrow* code from other programs (so you don't have to write it yourself) and to quickly assemble your program from blocks of code that already work. This means you can concentrate on the logic of your program and how it works - not on whether there is a problem positioning the cursor.

The following pages list many commonly used COMAL 0.14 procedures and functions that have been written in COMAL (and sent to COMAL Users Group) since the start of *COMAL Today*. They all are on *Today Disk #15*. To make your own library disk of procedures and functions, follow these steps for **each** one:

1) Type: NEW
2) Type: AUTO 9000
3) Type in the procedure or function
4) COMAL 0.14: Hit <return> to stop AUTO
5) Store to disk: LIST "NAME.PROC"

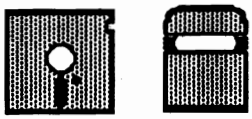
SPECIAL NOTES:

There are many more procedures and functions available for your library disk. Each issue of *COMAL Today* has provided new routines (also available on each *Today Disk*). We are including the **1520 Plotter** routines on *Today Disk #15* for your use. They are explained in *COMAL Today #7* on page 62. Another important set of routines are the **Disk Get** routines, originally from the *COMAL Handbook* Appendix D. They also are on *Today Disk #2*.

These routines are meant for COMAL 0.14 users. Some may be used by COMAL 2.0 users. However, use the **MERGE** command rather than **ENTER**, and it is not necessary to issue a **RENUM** command after it.

Finally, you can use the **listerine** program from *COMAL Today #14* to pull out procedures and functions from any of your programs. ■

Floating Point Error

by David Stidolph 

In the last issue of *COMAL Today*, we published a **POKE** that would change the method of drawing on the Hi-Res graphics screen. It flips the status of a pixel, rather than setting it (so that an on bit is turned off, and an off bit is turned on). As a demonstration for this, I wrote a short COMAL 2.0 graphics program that drew a series of eight boxes rotated about a point. I intended to draw boxes in one pass and erase them in the next. I was quite surprised to find that after several passes, some of the pixels never got erased. The group of boxes slowly floated up towards the top of the screen, leaving a trail behind.

The answer turned out to be quite simple. It seems that COMAL 2.0 carries out all its graphic functions using floating point numbers. A small roundoff error accumulated over a period of time, preventing the drawing from ending on the same point it started from.

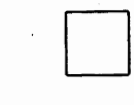
What was more surprising was that when I tried this in COMAL 0.14 (with a different **POKE** to change drawing to flipping), the pattern stayed in its original position. Is COMAL 0.14 better than COMAL 2.0 with floating point math? I don't know, but it makes for an interesting discussion. *[See page COMAL Today #15, page 23 for another example of the difference in the roundoff error between the different versions of COMAL.]*

COMAL 2.0 Program:

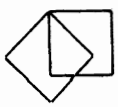
```
USE graphics // initialize graphics
graphicscreen(0)
moveto(160,100) // move to center
POKE $c462,$5d // flip mode to draw
//
REPEAT
  FOR y:=1 TO 8 DO
    box
    right(45)
  ENDFOR y
UNTIL TRUE=FALSE
//
PROC box
  FOR x:=1 TO 4 DO
    right(90)
    forward(50)
  ENDFOR x
ENDPROC box
```

COMAL 0.14 program:

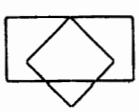
```
setgraphic (0)
moveto 160,100
hideturtle
poke 27669,93
//
repeat
 for y:=1 to 8 do
  box
  right (45)
 endfor y
until true=false
//
proc box
 for x:=1 to 4 do
  right (90)
  forward (50)
 endfor x
endproc box ■
```



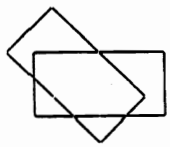
STEP 1



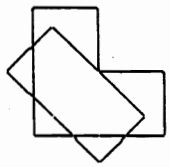
STEP 2



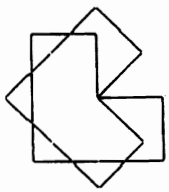
STEP 3



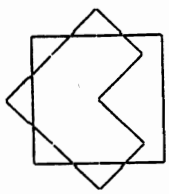
STEP 4



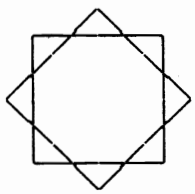
STEP 5



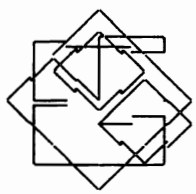
STEP 6



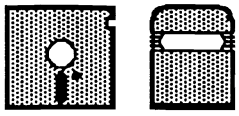
STEP 7



STEP 8



Calculate PI

by Jack Baldrige 

There's not really much reason to calculate the value of **PI** to any great accuracy. I've read somewhere that with a value to 25 digits, you could express the diameter of the earth to the nearest atom.

Still, before the advent of computers, somebody calculated the value of **PI** to over 500 places by mechanical means. The first time it was done with a computer was with Eniac in 1949. It calculated **PI** to 2037 places in 70 hours. Even the C64 does much better than that today. It took the C64 less than 14 hours to get 2500 places using COMAL.

Today, there is one practical use for these calculations. If you get a few hundred thousand digits on a new computer, you know that it's reliable (in integer operations, anyway). The latest such job I have figures on was a calculation in France on a CDC 6600. It got 500,000 digits (and checked them with a different algorithm) in 44 hours, 45 minutes. I know, however, that the same trick has been done with supercomputers such as the Cray, so heaven knows the latest number of digits.

The most common algorithm, the one I used, was developed by John Machin about 1700:

pi = 16*arctan(1/5)-4*arctan(1/239)

The arc tangent, from Gregory's series, is:

arctan(1/x) = 1/x-1/(3*x^3)+1/(5*x^5)-....

As you can imagine, the arctan series converges quickly when x=5 and goes like fury for x=239!

Admittedly, there isn't a very good reason to do a calculation like this, but at least, you get a result that's a little more accurate than you get with a pocket calculator. *Calculate'pi* is on both sides of *Today Disk #17*.

```
3.14159265358979323846264338327950288419716939937510582097494459
23078164062862089986280348253421170679821480865132823066470938446095505822317253594081284811174502841027019385211055596446229489549303819644288109756659334461284756482337867831652712019091456485669234603486104543266482133936072602491412737245870066063155881748815209209628292540917153643678925903600113305305488204665213841469519415116094330572703657595919530921861173819326117931051185480744623799627495673518857527248912279381830119491298336733624406566430860213949463952247371907021798609437027705392171762931767523846748184676694051320005681271452635608277857713427577896091736371787214684409012249534301465495853710507922796892589235420199561121290219608640344181598136297747713099605187072113499999983729780499510597317328160963185950244594553469083026425223082533446850352619311881710100031378387528865875332083814206171776691473035982534904287554687311595628638823537875937519577818577805321712268066130019278766111959092164201989380952572010654858632788659361533818279682303019520353018529689957736225994138912497217752834791315155748572424541506959508295331168617278558890750983817546374649393192550604009277016711390098488240128583616035637076601047101819429555961989467678374494482553797747268471040475346462080466842590694912933136770289891521047521620569660240580381501935112533824300355876402474964732639141992726042699227967823547816360093417216412199245863150302861829745557067498385054945885869269956909272107975093029553211653449872027559602364806654991198818347977535663698074265425278625518184175746728909777727938000816470600161452491921732172147723501414419735685481613611573525521334757418494684385233239073941433345477624168625189835694855620992192221842725502542568876717904946016534668049886272327917860857843838279679766814541009538837863609506800642251252051173929848960841284886269456042419652850222106611863067442786220391949450471237137869609563643719172874677646575739624138908658326459958133904780275901
```

```
dim q$ of 1 // 2.0 users must add a () after
print chr$(147) // array names as parameters
print "   high precision pi calculations"
print
input " minimum number of digits? ": digit#
max#:=digit# div 4+2
digit#:=max#*4-4; array'size#:=max#
hcopy#:=false
input " output to printer? n"+chr$(157): q$
if q$="Y" or q$="y" then hcopy#:=true
setzero
header
dim pi'array#(max#), partial#(max#)
dim n$ of digit#, b$ of 10
b$:="0123456789"
series(pi'array#,5,max#) // 2.0 add ()
multiply(pi'array#,4,max#) // 2.0 add ()
series(partial#,239,max#) // 2.0 add ()
subtract(max#,partial#,pi'array#) // 2.0 add ()
multiply(pi'array#,4,max#) // 2.0 add ()
print
if hcopy# then
 select output "lp:"
 header
endif
print'pi(max#,pi'array#,n$,b$) // 2.0 add ()
j:=int(seconds*100+.5)/100
print "time is";j;"seconds"
select output "ds:"
end
//
proc series(ref array#(),f,max#) closed
 dim power#(max#), temp#(max#)
 power#(max#):=1000; array'size#:=max#
 divide(power#,f,array'size#,max#) //2.0 add ()
 sign:=1; divisor:=3; count:=0
 for k:=1 to max# do array#(k):=power#(k)
 while array'size#>0 do
```

more»

Calculate PI - continued

```
  count:+1
  divide(power#,f*f,array'size#,max#)//2.0 add ()
  sign:=-sign
  for j#:=1 to max# do temp#(j#):=power#(j#)
  divide(temp#,divisor,array'size#,max#)//2.0add()
  divisor:+2
  if sign>0 then
   add(array'size#,temp#,array#) // 2.0 add ()
  else
   subtract(array'size#,temp#,array#)//2.0 add ()
  endif
 endwhile
endproc series
//
proc divide(ref array#(),divisor,ref array'size#
 ,max#) closed // wrap line
 remainder:=0; array'size#:=max#
 while array#(array'size#)=0 do
  array'size#:-1
  if array'size#=0 then return
 endwhile
 for i:=array'size# to 1 step -1 do
  term:=10000*remainder+array#(i)
  array#(i):=term div divisor
  remainder:=term mod divisor
 endfor i
 if array#(array'size#)=0 then array'size#:-1
endproc divide
//
proc add(ref array'size#,ref tarray#(),ref
 sarray#()) closed // wrap line
 carry:=0
 for k#:=1 to array'size# do
  sum:=tarray#(k#)+sarray#(k#)+carry
  sarray#(k#):=sum mod 10000
  carry:=sum div 10000
 endfor k#
 if carry then sarray#(array'size#+1):+carry
endproc add
//
proc subtract(ref array'size#,ref termarray#(),ref
 diffarray#()) closed // wrap line
 borrow:=0
 for k#:=1 to array'size# do
  diff:=diffarray#(k#)-termarray#(k#)+borrow
  borrow:=diff div 10000
  diffarray#(k#):=diff mod 10000
 endfor k#
 if borrow then diffarray#(array'size#+1):-1
endproc subtract
//
proc multiply(ref array#(),factor#,max#) closed
 carry:=0
 for j#:=1 to max# do
  product:=factor#*array#(j#)+carry
  array#(j#):=product mod 10000
  carry:=product div 10000
 endfor j#
endproc multiply
//
proc print'pi(max#,ref array#(),ref n$,ref b$
 ) closed // wrap line
 dim p$ of 4
 print
 for k#:=max# to 2 step -1 do
  string(array#(k#),p$,b$)
  n$:=n$+p$
 endfor k#
 print "pi equals";n$
 for k#:=1 to 3 do print
endproc print'pi
//
proc header
 print " calculating pi to";digit#;"digits..."
endproc header
//
proc string(ref n#,ref p$,ref b$) closed
 hi:=n# div 100; lo:=n# mod 100
 p$:=b$(hi div 10+1); p$:=p$+b$(hi mod 10+1)
 p$:=p$+b$(lo div 10+1); p$:=p$+b$(lo mod 10+1)
endproc string
//
func seconds closed
 j:=256*256*peek(160)+256*peek(161)+peek(162)
 return j/60
endfunc seconds
//
proc setzero closed
 poke 160,0
 poke 161,0
 poke 162,0
endproc setzero ■
```

Guest Editorial

by Robert Ross

Reading the article in *COMAL Today #15* on the 1986 COMAL Standards Meeting, I realized that no one invited me. Boy, am I miffed. No matter that I probably would have declined an invitation with some all-wet excuse such as Denmark is a long way to swim and plane tickets cost money. But if I had been invited and if I had shown up and could listen dansk, I'm sure I would have enjoyed the debate of *"What value should be returned by the expression a$ IN b$ when a$ is the empty string?"* If I spoke dansk (I rarely USE dansk), I might have even joined in. After all, I was co-captain of the debate team my senior year in high school. No matter that the coach didn't actually let me debate that year. Encouraged by David Stidolph's question in *COMAL Today #16*, *"...what should "" IN "abc" return...,"* here goes.

Obviously, the expression should refuse all values that are incorrect and complain about poor quality control. Some of the confusion in this matter occurs because the operator in the expression is a real *smoothie*: it attempts to provide the answer to several questions at the same time. One has a **TRUE/FALSE** answer: is **a$** a substring of **b$**? Another is: what is the character position in **b$** of the first character of the first occurrence of **a$** in **b$**, provided that **a$** is a substring of **b$**?

A function to answer only the first question could be implemented several ways. The function could always return 0 for **FALSE** and 1 for **TRUE**. Or perhaps the function could return a count of the number of times **a$** occurs in **b$**, such as 3 for **"a" IN "abacad"**. But still the question of **a$=""** remains. The practical person might ask, *"Why do a test on something that is always true? You may as well program like:"*

```
IF a$="" AND NOT a$ IN b$ THEN
  REPORT 51,"system error in truth, forsooth."
ENDIF
```

On the other hand, the impractical person, especially one who understood set theory, might respond, *"The empty string is a substring of every string and must be reported as such: Tell it like it is!"* If this isolated substring function were to be a keyword, I would vote for an operator keyword called:

is'not'the'empty'string'and'is'a'substring'of

but with a much shorter name, such as..such as?

The empty string would also present a problem for a function to answer just the **position'of'a'substring'in** part of **IN. More people would affirm that "" is a substring of every string than would be willing to assign a character position to it.** If you allow **LEN(b$)+1** why not **LEN(b$)+2** also, not for the function value, which represents only one character position, presumably the first, but as the position of another appended ""? If two, why not an infinite number of empty strings appended to the end of every string? And a similar number preceding the string! What then of **LEN(b$)**? Should it always return the dimensioned length of **b$**? Or how about *overflow* every time it's used? No! Assigning a character position to "" is at least as bad, perhaps worse, than failing to report it as a substring.

The **position'of'a'substring'in** function also has a problem when the first string is not a substring of the second string, which is probably why it was originally combined with the **TRUE/FALSE** question for the operator **IN**: what character position can be returned if the position is requested when it doesn't exist? A *fictional position* error could be reported, but sometimes it is preferable to return a value that can be tested rather than to generate an error. Recognizing that **IN** is a combination of functions, I vote for the hodgepodge I have implemented as a function **position'in**.

more»

```
func position'in (a$,b$) closed
  if a$="" then
    return 0
  elif not a$ in b$
    return false
  else
    return a$ in b$
  endif
endfunc position'in
```

It returns zero as the position of the empty string in the second string; this is just as valid as LEN(b$)+1. It will return **FALSE** if the first string is not a substring of the second. It will return the character position in the second string of the first character of the first occurrence of the first string in the second when the first string is a substring of the second. By inspection of the function, it should be obvious that **IN** is a better name for this operation than **position'in**.

If this function doesn't replace the current definition of **IN**, my compromise vote is for a possibility that wasn't on the list in the article about the Standards meeting: "" IN "abc" could return -1. Negative values test true, to satisfy theoretical people, and in situations like:

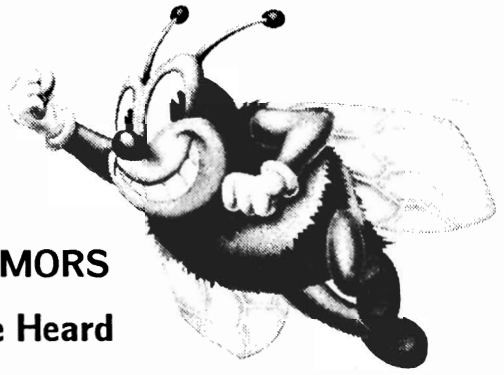
```
REPEAT
 a$=KEY$
UNTIL (a$ IN b$)>0
```

the last line is easier to type than:

```
until a$ in b$ and (a$ in b$) < len(b$)+1
```

Note that both versions obscure part of what is being tested and leave something to be desired in ease of readability and understanding.

Much thought has gone into COMAL standards. And the continuing discussions may be long, but they don't have to be acrimonious: COMAL lets you easily define a new function or re-define an old one to get exactly what you need. ■



RUMORS We Heard

by Captain "*Buzz*" COMAL

Mytech called to tell us that they have corrected the bugs in their **IBM PC COMAL** and expect to release the **final in August 1987**. Expect the selling price to be **under $100**.

Mytech also said they have a **COMAL up and running on an Amiga**. They are testing it now for release in **September 1987**. They expect the price to be **under $200**.

Last issue we told you about the **CP/M COMAL RUNTIME "compiler"** system. It has been **shipping for over a month now**. We even have a special on it (see page 77). Now it seems that a **cross-compiler** is under way for the IBM and Amiga. **Apple COMAL** is designed with a compiler in mind as a **future option**.

The **MacIntosh COMAL** project may be revived soon. We have a test version running on a Mac here, and hope it will be finalized. But an **Atari ST** COMAL may be out before that!

COMAL was originally designed **specifically for use by schools**. By the end of the year it should be running on most of the computers used by schools: PET, Apple II, C64, C128, CP/M, IBM, Amiga, Atari, and MacIntosh.

Meanwhile, we hear that Mytech is working on a special version to **run on a college mainframe computer**. Soon students can learn COMAL on an Apple II or IBM PC **at school**, and run the same programs on their C64 or C128 **at home**. Later, **in college**, the programs will run on the schools mainframe. ■

DISKS: (Buy one for $7.95 - get one FREE)

Take one **free** COMAL disk for each item over $5 you buy! Buy Beginning COMAL, get a free disk. Buy the Power Box, get a free disk. Even buy a disk, get another disk free! To buy a disk from the list, write the word **buy** in the [ ] box. To pick a free disk, write the word **free** in the [ ] box. Subscriber price is just **$7.95 per disk**. **Choose from the disks below:**

[ ] CP/M COMAL Demo disk (Osborne format)
[ ] Beginning COMAL disk §
[ ] Foundations with COMAL disk §
[ ] COMAL Handbook disk §
[ ] Games Disk #1 (0.14 & 2.0)
[ ] Modem Disk (0.14 & 2.0)
[ ] Article text files disk
[ ] Today Disk (one disk side--circle choices):
1 2 3 4 5 20

§ = these disks assume you have the book

0.14 Disks:

[ ] Data Base Disk 0.14
[ ] Tutorial Disk
[ ] Auto Run Demo Disk
[ ] Paradise Disk
[ ] Best of COMAL
[ ] Utility Disk 1
[ ] Slide Show disk (circle which): 1 2
[ ] Spanish COMAL
[ ] User Group 0.14 disks (circle numbers):
1 2 3 4 5 6 7 8 9 10 12

2.0 Disks:

[ ] Data Base Disk 2.0
[ ] Superchip programs disk
[ ] Read & Run
[ ] Math & Science
[ ] Cart Demo (circle which): 1 2 3 4
[ ] 2.0 User disks (circle choices):
11 13 14 15

Note: each disk below counts as two free disks, but still costs only $7.95 to buy:

[ ] Bricks 0.14 Tutorial (2 disks)
[ ] COMAL Today INDEX disk (2 disks)
[ ] 2.0 Typing disk (2 disks)
[ ] Today Disk (2 sides. 0.14 front/2.0 back):
6 7 8 9 10 11 12 13 14 15 16 17 18 19

Note: Some disks may be supplied on the back side of another disk. Disk format is Commodore 1541 unless specified otherwise. We replace any defective disk at no charge if you return the disk with a note explaining what is wrong with it. Some disks are being reduplicated and appropriately relabeled. Please do not give out copies of our disks.

SPECIAL DISKS & DISK SETS:

[ ] $10.95 **Sprite Pak**
Two disk set. Huge collection of sprite images, sprite editors, viewers, and other sprite programs. For 0.14 and 2.0.

[ ] $10.95 **Font Pak**
Three disk set. Collection of many different character sets (fonts) for use with 0.14 and 2.0 including special font editors!

[ ] $10.95 **Graphics Pak**
Five disk set. Picture heaven. Includes Slide Show, Picture Compactor, Graphics Editor and lots of pictures (normal and compacted)

[ ] $29.95 **Sprite, Font & Graphics Pak**
All ten disks mentioned above!

[ ] $10.95 **C128 CP/M Graphics**
Graphics package on disk for use with CP/M COMAL on the C128. Includes turtle graphics and preliminary Font package.

[ ] $10.95 **Guitar Disks**
Three 0.14 disk set. Teaches guitar by playing songs while displaying the chords and words.

[ ] $10.95 **Cart Demo Disks**
Four disks full of programs demonstrating the many features of the C64 2.0 cartridge.

[ ] $10.95 **Shareware Disks**
Three disk set. Includes a full HazMat system (Hazardous Materials), an Expert System, Finger Print system, Traffic Calc, and a BBS program.

[ ] $10.95 **Superchip On Disk**
All the commands of Super Chip (but not the Auto Start feature) disk loaded.

[ ] $10.95 **Super Chip Source Code**
Full source code with minimal comments. Customize your own Super Chip. Add commands. Remove the ones you don't need.

[ ] $10.95 **2.0 User Group Disks**
Four disks set for the C64 COMAL cart.

[ ] $24.95 **0.14 User Group Disks**
Twelve disks (User Group disk 9 is a newsletter on disk system, double disk).

[ ] $24.95 **European Disk Set**
Twelve 2.0 disk set. Find out what COMALites in Europe are doing.